



WORCESTER POLYTECHNIC INSTITUTE

SaunterNot

The design, development, implementation, and testing of a device for detecting patients who wander off at nursing home facilities which alerts the nursing staff in a quick and efficient way.

A Major Qualifying Project work program plan to be submitted to the faculty of Worcester Polytechnic Institute's Department of Electrical and Computer Engineering in partial fulfillment of the requirements for the Degree of Bachelor of Science.

Submitted by:

William Brooks _____

Edmund Massa _____

James Medeiros _____

Stacey Mohr _____

Submitted to:

Professor Labonté _____

Professor Michalson _____

Abstract

With Nursing home facilities enduring staff shortages, the SaunterNot system would provide a safe and efficient way of alerting the staff if a patient suffering from a neurological disease is attempting to exit the building. Each patient who is examined and found likely to be at risk of wandering out of the building into what may become hazardous situations will be asked to wear a transponder tag. Each tag is encoded with an 8-bit identification number which is cross-referenced to their identity. When passing through a door that exits the building, a reader placed at the door will receive a signal from the transponder and trigger an alarm to alert the staff of the resident exiting the building.

Table of Contents

Abstract.....	2
1.0 Introduction.....	8
1.1 Competition Analysis.....	9
1.2 Enhanced Features	10
2.0 Design Requirements	11
2.1 Transponder (tag).....	11
2.2 Receiver	12
2.3 Reader	12
2.4 Value Analysis	13
2.4.1 Requirement Weighting.....	13
2.4.2 Cost Analysis	13
2.5 Artist Concept Drawing	16
3.0 Market Research	17
3.1 Global Positioning Systems (GPS).....	17
3.1.1 Indoor Geolocation Systems	18
3.2 Wireless Technology	19
3.3 Motion Detection	20
3.3.1 Pyroelectric Sensors.....	21
3.4 Radio Frequency Identification (RFID).....	21
3.4.1 Active RFID.....	22
3.4.2 Passive RFID	22
3.4.3 Design of a Transponder System	25
3.5 E-ZPass	26
3.6 System Analysis.....	27
4.0 Initial Designs	29
4.1 Design I.....	29
4.1.1 Block Diagram.....	29
4.1.2 Analysis.....	30
4.2 Design II.....	31
4.2.1 Low Frequency (LF) – 134kHz	32
4.2.2 High Frequency (HF) - 13.54MHz	34
4.2.3 Ultra High Frequency (UHF) - 900MHz	36

4.2.4 Value Analysis	38
5.0 System Design Requirements	41
5.1 Aloha Transmission Encoding.....	41
5.2 Encoder/Decoder Pairs.....	42
5.3 Power Analysis	42
5.4 Reading Range	44
6.0 Functional Entities	46
6.1 Block Diagram	46
6.1.1 Transponder	46
6.1.2 Receiver	55
6.1.3 Reader	57
6.1.4 Computer Interface	62
7.0 Design Testing	66
7.1 Transponder	66
7.2 Receiver	70
7.3 Central Nurses Station	71
7.4 Analysis.....	71
8.0 Conclusion	73
8.1 Finances	73
9.0 Recommendations.....	77
9.1 Device Marketability	77
9.2 Alternative Design Uses	78
9.3 Future Projects	80
Appendix A: Interview with Cie Galloway	81
Appendix B: Resident Rights Category	83
Appendix C: Transponder Code	84
PIC Code.....	84
Appendix D: Spartan Code	87
Parallel_rx.....	87
Appendix E: Spartan Code.....	88
Hex_Driver2w2.....	88
Appendix F: Spartan Code.....	90
comp.....	90
Appendix G: Spartan Code	93

serial_tx2.....	93
Appendix H: Spartan Code	97
serial_tx2.....	97
Appendix I: Computer Code	99
Main.cpp	99
Appendix J: Computer Code.....	103
serialport.h	103
serialport.cpp.....	106
Appendix K: Computer Code	121
CSpreadSheet.h.....	121
Appendix L: Computer Code.....	147
SN.h	147
Datasheets	150
Bill of Materials	151
Works Cited	152

Table of Figures

Figure 1: WanderGuard® System for Hospital Use	9
Figure 2: Return on Investment	15
Figure 3: Artist Concept of SaunterNot System	16
Figure 4: Trilateration	17
Figure 5: Geolocation Triangulation.....	19
Figure 6: Fresnel Lens and Plano Convex lens.....	21
Figure 7: Design I - Block Diagram	30
Figure 8: Low Frequency Design Option	33
Figure 9: Low Frequency Design Costs.....	34
Figure 10: High Frequency Design Option.....	35
Figure 11: High Frequency Design Costs	35
Figure 12: Ultra High Frequency Design Option	37
Figure 13: Ultra High Frequency Design Cost	37
Figure 14: Design III - Block Diagram.....	46
Figure 15: Transponder Block Diagram	47
Figure 16: Boost Converter Configuration per DataSheet.....	48
Figure 17: Boost Converter Configuration per Design III.....	48
Figure 18: Internal Clock of the PIC.....	51
Figure 19: USART Block Diagram	52
Figure 20: Digital Sequence between Clock Cycles and USART	54
Figure 21: PIC Schematic	54
Figure 22: Output of RC6/TX pin and Receiver.....	55
Figure 23: Tx/Rx Pair	57
Figure 24: Receiver to Reader Block Diagram.....	57
Figure 25: Reader Flow Diagram	58
Figure 26: Reader Block Diagram	59
Figure 27: CONTROL Module.....	61
Figure 28: RS232 Data Byte Sequence.....	63
Figure 29: Computer Pop-Up Alarm	65
Figure 30: Design III - Detailed Block Diagram	66
Figure 31: Boost Converter Output Voltage Ripple	67
Figure 32: Data Out from PIC Microcontroller	68
Figure 33: Bit Length of One Bit of PIC Output	68
Figure 34: Transmitter Input/ Receiver Output	69
Figure 35: Transmitter and Receiver Time Delays.....	69
Figure 36: Receiver Output Bit Length	70
Figure 37: Tx, Rx and Reader Clock Waveforms.....	71
Figure 38: Parts Order Form	74
Figure 39: November 27, 2007 Order Form	75
Figure 40: January 11, 2008 Order Form.....	76
Figure 41: April 2, 2008 Order Form.....	76
Figure 42: WanderGuard Transmitter Device	82

List of Tables

Table 1: System Analysis.....	27
Table 2: Design II - System Value Analysis.....	38
Table 3: Design II - Antenna Value Analysis.....	39
Table 4: Design II - Value Analysis	39
Table 5: Battery Considerations.....	43
Table 6: Options for PIC microcontroller.....	50
Table 7: Baud Rates and Error Margins for PIC.....	53
Table 8: Distance Tests.....	72

1.0 Introduction

There are approximately 17,000 nursing homes in the United States which provide care for over 1.5 million patients over the age of 65 according to a 1997 National Nursing Home Survey [1]. These patients receive all forms of care from being bathed to hospice care. Roughly 70% of nursing home residents have some form of neurodegenerative diseases. More specifically, 42% of patients have dementia and 12% have some other form of neurodegenerative disorders. Nursing homes can be rather expensive at approximately \$56,000 for a year; averaging to \$153.00 per day. Medicare and Medicaid assist in the costs, however it is still a considerable expense to provide care for the elderly [2].

Across the nation, nursing homes are about 50% understaffed; decreasing the quality care of the patients in these nursing home facilities [3]. One way to mitigate this issue is to invent a device that will help nurses keep track of patients. For instance, the SaunterNot System would be a device capable of alerting the nursing staff when patients try to wander outside the nursing home facility. The objective of this system is to allow staff and visitors to come and go without using a keypad to open the doors, but when a patient wanders off an alarm will sound and the staff will be notified who is attempting to leave and what door they are exiting. This device will help to ease the nurses worries on the whereabouts of various patients allowing them to focus their efforts on patients who require more care.

Knowing a loved one who suffers from dementia, Alzheimer's or other neurodegenerative diseases can be emotionally draining. Individuals suffering from a neurological disease have deteriorating cognitive skills beginning with a lapsing memory. As the disease progresses, individuals try to retreat to a time and place they remember and are known to attempt to leave unfamiliar areas; such as nursing homes. The SaunterNot system aims to provide a sense of security to the families of those suffering from neurodegenerative diseases by keeping the nursing staff well informed of any patients who attempt to wander out the building, so they know their loved ones are safe.

The mission of this project is to design, develop, implement, and test a device for detecting patients who wander off at nursing home facilities by alerting the nursing staff at these facilities in a quick and efficient way. The ability to locate and know who is trying to wander off will provide crucial information to the nursing staff so they can know who wanders the most and who they will need to watch more carefully. Families should feel comfort and security from the

SaunterNot system because their relatives will be in a safe and protected environment. The SaunterNot system will help the nursing staff, the families, and the patients due to the system's alerting and identification capabilities.

1.1 Competition Analysis

The WanderGuard® System is a device currently used in nursing homes to detect when a resident is wandering through an exit door. This system is appropriate for use with patients suffering from neurodegenerative diseases or those that present a flight risk. The WanderGuard® System is available for hospitals; Figure 1 [4], long-term care, or for use in a residential community.



Figure 1: WanderGuard® System for Hospital Use

If the individual wearing the bracelet attempts to walk out a door that has the WanderGuard® System installed, as the bracelet crosses the infrared beam an alarm is sent to the device located at the nurse's station and the door monitor sounds an alarm at the exit location. This provides anyone within earshot of the door a warning that someone who is not suppose to be exiting the building is attempting to do so, and the alarm at the nurses' station warns those not within immediate earshot of the door.

Regardless of the environment application of the WanderGuard® System, its basic function remains the same. If an individual is assessed and deemed a possible flight risk, they are to wear a bracelet around their ankle at all times. The bracelet is "small and lightweight" has a "waterproof case" and is fastened with a "durable nylon-reinforced band" [4]. The removal of the bracelet only comes at times of bathing and when exiting the facility with assistance until the time comes when the patient is no longer deemed at risk of wandering.

Exit doors that pose a threat throughout the area are equipped with a door monitor that emits an infrared beam from approximately two 1 foot tall pillars facing each other.

1.2 Enhanced Features

The SaunterNot system aims to not only achieve the same functionality as the WanderGuard®, but will also enhance several features. After conducting an interview with the interim nursing director, Cie Galloway (see Appendix A) at Parson's Hill Rehabilitation Center, valuable first hand information in regards to the current products performance was obtained. Patients were found to dislike the bulky bracelets and often attempted to and were successful in taking off the bracelets. When patients who had the bracelets on began exiting a doorway, a loud alarm would sound agitating the patient. It would then take some time for the staff to calm the patient and help return them to their room. Other patients with the bracelets still placed around their ankles were observed to open the exit door and step over the pillars thus avoiding the infrared beam and could leave without detection. When patients were detected, alarms would sound at the door and an alert would be sent to the nurses' station, however the staff never knew who it was that just left.

In order to improve on the current products functionality, several changes and enhancements can be made. For instance, the SaunterNot system will use a smaller tag that patients will wear increasing their comfort level. With a small tag, this could be placed discretely in or on an item the patient uses or wears daily; such as a watch, so that they will not have the constant desire to remove the device, or feel awkward with its appearance. When the patient sets off the alarm by exiting the doorway, rather than sounding an alarm, a silent alarm of flashing lights above the door or a unique song that the staff would recognize would avoid agitating the patient. To avoid the possibility of the patients evading the system; such as stepping over the infrared beam, a form of blanket coverage over the entire doorway would be most ideal to ensure that anyone with a tag who exits the door is detected. Finally, and what is also deemed as one of the most desirable enhanced functionalities the SaunterNot system would have over its competitors is its ability to identify the patient who has left an exit door. Knowing what patient they are looking for would considerably help the staff determine where the individual may have headed based on patients history and tendencies.

2.0 Design Requirements

Our requirements for this project have been chosen to meet suggestions from people with experience working with this type of system in its actual environment and were also based from research in regards to the competition currently in the marketplace. One individual who proved to be a profuse resource on this matter was Mrs. Cie Galloway, interim Director of Nursing at Parson's Hill Rehabilitation Center in Worcester, MA (Appendix A). During our meeting, Mrs. Galloway provided insight into the current systems being used at a variety of nursing facilities through her experiences with nine different rehabilitation centers. Mrs. Galloway was also able to provide information about what she and the staff feel is the most important feature that is currently lacking in their system; person identification.

Based on our conversation with Mrs. Galloway and research conducted, a number of requirements were synthesized. The requirements have been broken up based on the physical pieces of equipment to which they correspond.

2.1 Transponder (tag)

The device should:

Size: be approximately 1" x 1" x 0.5"

Cost: cost between \$10 to \$20

Lifetime: have a lifespan of at least 1 year

Durability: be water resistant to any accidental immersion into liquid, or liquid spilled onto it. It should also be able to functionally survive a drop from 6' onto a concrete surface.

Weight: weigh less than 15 grams

Functionality: communicate a unique transponder ID to the receiver using RFID

Optional Requirements: It should be able to monitor an individual's heart rate and display that information to a nurse.

2.2 Receiver

The device should:

Size: be no larger than 6" x 2" x 1"

Cost: cost less than \$250

Lifetime: have a lifespan of at least 10 years

Durability: be water resistant to any accidental immersion into liquid lasting less than 10 seconds, or liquid spilled onto it. It should also be able to functionally survive a drop from 6' onto a concrete surface

Weight: weigh between 1 to 2 lbs

Power: be powered by a standard 120 VAC outlet, either to be hooked directly to the electrical system of the building, or into a standard outlet.

Functionality: be able to identify up to 10 unique transponder signals within its detection range. It should also be able to alert nurses at a centralized station that a specific individual is attempting to move through an exit door. It should also have an LED flash when someone attempts to move through an exit door.

2.3 Reader

The device should:

Size: be no larger than 6" x 8" x 3"

Cost: cost less than \$250

Lifetime: have a lifespan of at least 10 years

Durability: be water resistant to any accidental immersion into liquid lasting less than 10 seconds, or liquid spilled onto it. It should also be able to functionally survive a drop from 3' onto a concrete surface.

Weight: weigh between 1 lb and 3 lbs

Power: be able to be powered by a standard 120 VAC outlet

Functionality: be able to receive signals from the receiver, and accurately identify the individual and doorway sending the signal at no less than a 95% yield. It should also be able to communicate with a computer database to access an individual's information when their transponder triggers the system, and communicate the individual's name to the nursing staff.

2.4 Value Analysis

After defining several requirements, a weighting system of what is essential to the needs of the SaunterNot device was developed. A cost analysis was also performed in order to determine the best solution for the design.

2.4.1 Requirement Weighting

The requirements are weighted in order of most important to least important:

- 1) Functionality
- 2) Lifetime
- 3) Durability
- 4) Cost
- 5) Weight
- 6) Size

This order was chosen because it was felt that having a product that works as it is intended, is durable, and is long lasting (all of which can fall under the category as dependable) is more important than making a cheap, small, and light product that does not provide all of the required functionality.

If our product were not dependable, this could directly impact the safety of its users. If the transponder fails to alarm the receiver as a patient exits the door, the patient becomes at risk of wandering into traffic or other places that could be dangerous. Also, if the product does not properly identify the individual who has left, the staff may look for the wrong patient and could call and inform the wrong family on the person's disappearance. This type of mistake would not only hinder the search for the individual but would emotionally disturb the family.

2.4.2 Cost Analysis

When making a product, one must consider exactly how much the product will cost and the cost of production. Other things to consider with a new product are initial costs to start up a business, making a product, and the turnaround investment time. For the sake of simplicity, we are going to try to start as inexpensive as possible. This means renting equipment and building the product out of a basement as opposed to a manufacturing facility. This of course will be a rough estimate, and there are likely various minute factors overlooked.

2.4.2.1 Initial Investment

Research on the parts that potentially could be used to produce the transponder and the receiver gave us an estimated figure of how much they would cost. Parts for the transponder would cost about \$10, and the receiver would cost around \$200. In order to help decrease production costs, the SaunterNot system could be manufactured in a basement while renting equipment at \$1,000 per month. Using four engineers who wish to get paid approximately \$50,000 dollars a year for this project will raise our cost to \$16,667 per month. If we want to produce 100 receivers a month and sell all of them, that would cost about \$20,000. For each receiver sold, if five transponders are sold that would raise the cost to \$5,000 for a total of \$25,000. Taking this into consideration, the starting cost would be about \$43,000 every month. Additional costs such as advertising may apply as well, resulting in a final total of \$45,000 per month.

2.4.2.2 Return on Investment

In order to see a return on investment each unit would have to be sold at \$300 per receiver and \$25 per transponder. Ideally, at least 150 receivers and 750 transponders would be sold per month. Given that it would take several months until the product is on the market, we would start deploying units approximately three months after the production day. Figure 2 depicts how long it will take to break even where cost and profit intersect.

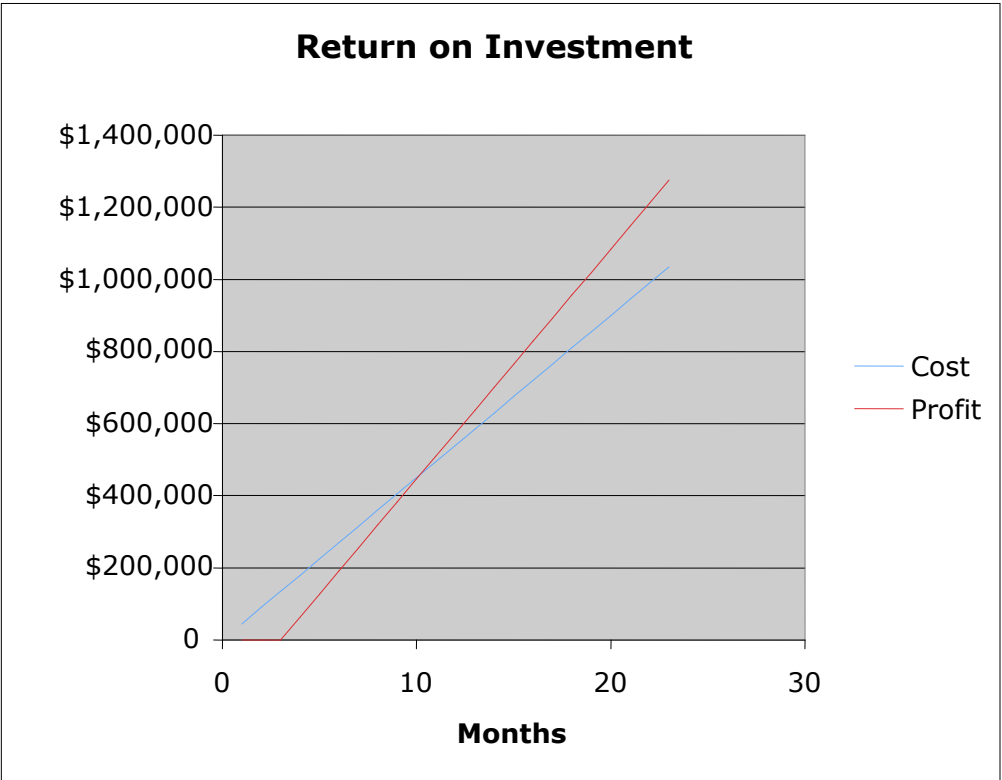


Figure 2: Return on Investment

According to Figure 2 cost would meet profits after 11 months. After the first 11 months we would start making a profit that would grow exponentially, allowing us to expand and increase production.

2.5 Artist Concept Drawing

Figure 3 represents an artist concept of the system. The system will consist of the receiver which is mounted on the mantle of a door. The receiver will send out a signal monitoring the door area. A transponder will be worn by the patient, and when the transponder receives the signal of the receiver, the transponder will send a signal containing identity information of the patient back to the receiver. This will occur when the patient steps through the door, and thus entering the receiver's monitoring field. The receiver will then send the patient's identity information to a nurse's station where the nurses will be alerted that patient 'John Smith' is wandering through an exit door.

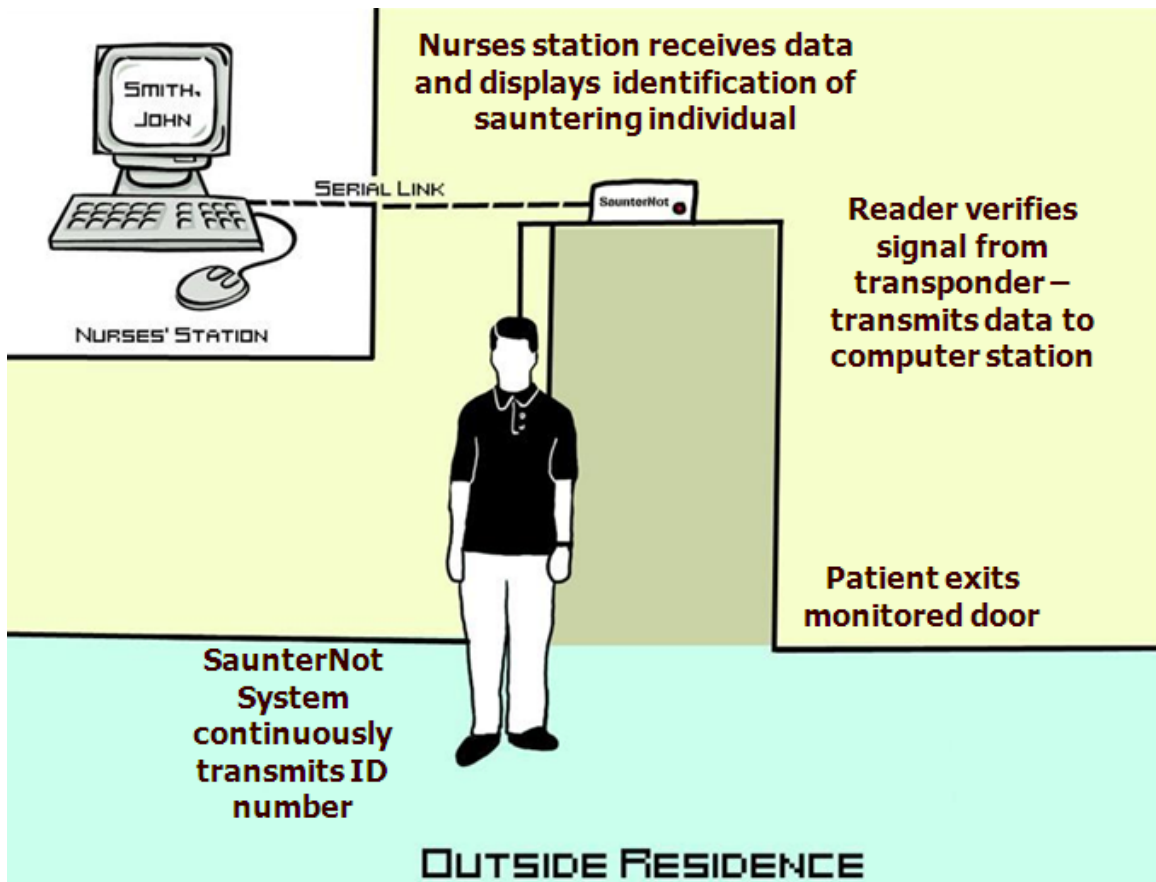


Figure 3: Artist Concept of SaunterNot System

3.0 Market Research

In order to design, develop, and implement a system that is capable of alarming when an individual wearing a tag or transponder breaches the field, a general knowledge of current architectural options is imperative. When taking into consideration the various tasks the system will need to accomplish, research on systems currently performing similar tasks is the first step to a system integration approach. In this section, systems such as Global Positioning Systems (GPS), Wireless Technologies, Motion Detection, and RFID are discussed. General background research on current systems and their applications will play an essential role to the design of the SaunterNot system.

3.1 Global Positioning Systems (GPS)

Beginning in the 1960's, the United States Navy and Air Force developed a satellite based positioning system as a means to accurately provide positioning and navigation support. In 1973, an organization was developed and given the name NAVSTAR in which representatives from the United States military services worked together to form a “comprehensive system to provide accurate data on position, velocity, and time to both military and civilian users” [5]. This system was referred to as Global Positioning System (GPS).

GPS is based off trilateration from several of 28 satellites orbiting the Earth. A GPS receiver is able to acquire signals from at least four of these 28 satellites. Using the distances calculated by the satellites to the receiver, the system can infer the position of the receiver.

The basic formula used for GPS navigation is $distance = rate \times time$. Knowing how fast the receiver is traveling and its relative location to a known stationary object, the system can calculate how long it would take for the receiver to arrive at its destination. The same methods apply to a stationary receiver in determining its location as shown in Figure 4 [6]. If it is known that the

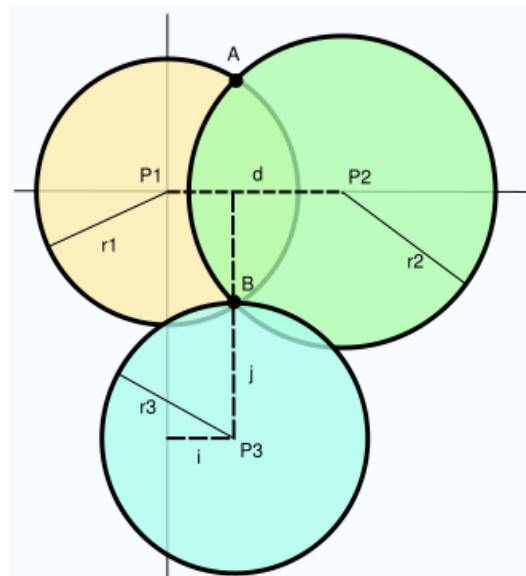


Figure 4: Trilateration

receiver is a distance r_1 from a relative object, it is known that the receiver must be within the radius of the object by a distance of r_1 . In order to further narrow down where the receiver is, compare its distance to another relative stationary object, being r_2 . With this information combined, the receiver is known to be in an area in which r_1 and r_2 overlap. A third measurement is needed to more precisely define the location of the receiver. By calculating the distance from a third relative stationary object, r_3 it can be concluded that the receiver is located at the point where all three of these radii intersect, B [6].

In order for this to work successfully, there must be a clear line of sight between the receiver and the satellites. GPS works ideally for outdoor applications such as surveying, transportation, or military use. However, difficulties arise when attempting to apply GPS to indoor applications.

Given that signals from the satellites are required to reach the receiver in order for the system to be successful, working indoors with a GPS system proposes the issues of multipath and signal interference. Multipath occurs as a result of the radio signals reaching the receiver by two or more paths with different phases. This effect has the ability to cause destructive interference and phase shifting of the signal which can deceive the system. Causes of multipath include the interference of ionospheric reflection and refraction, and the reflection from objects on land such as mountains and buildings. However, these interferences appear less in moving objects since when the receiver is in motion, the reflected signals will fail to converge leaving only the direct signals to feed back accurate solutions.

3.1.1 Indoor Geolocation Systems

GPS provides great accuracy for outdoor applications; however, due to the decreased line of sight of the signal, GPS is not as successful as an indoor application. This has led to a need for indoor geolocation systems to fill the void GPS leaves behind. Indoor geolocation has grown because of the need to track precisely where people or objects are indoors in both civilian and military applications. In most applications the system consists of an RF tag attached to a person or object, and several fixed stations to determine the approximate location of the tagged person or item. This is done by communicating with the tag and performing calculations.

One technology used for indoor geolocation systems is indoor GPS. Outdoor GPS does not work well indoors because of the “signal strength is too low to penetrate buildings” [7]. Indoor GPS uses a navigation signal produced by several “pseudolites (pseudo-satellites),” which generates a GPS-like signal. Since the pseudolites have a similar signal to GPS, indoor GPS receivers can be created with few alterations. The system observes reference receivers, and for successful navigation at least four pseudolites must be visible [7].

Indoor geolocation systems can use a variety of metrics including received signal strength (RSS). In RSS systems the receiver attains the distances from three fixed stations in the form of a vector. Each of these distances makes a circle, and combining these circles (using triangulation) reveals the location of the mobile station. Path loss of the building must be combined with the distance vectors gained from the fixed stations. This process can be seen in Figure 5 [7].

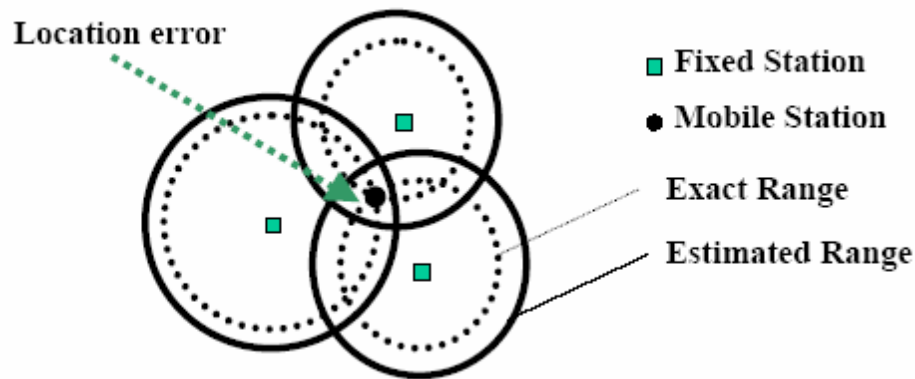


Figure 5: Geolocation Triangulation

3.2 Wireless Technology

Fixed, wireless, and internet communications have experienced their most rapid growth spurt within the past decade. The more common wireless systems currently being used include Wireless Area Networks (WANs), Local Area Networks (LANs), Personal Area Networks (PANs), and Wireless Local Area Networks (WLANs).

The wireless area network is a system commonly used for international coverage. One common device used within this system is a cellular phone. Local area networks are wired networks used within areas of 100 meters. Such applications are used in offices or small college campuses. Personal area networks operate in an even smaller area of approximately 10 meters. This type of network is used primarily for Bluetooth applications. The last commonly used

networking system is a wireless local area network. This type of network is similar to that of the local area network, however it operates without wires. A wireless local area network is commonly used in small businesses or residential homes for internet access.

The different systems operate in either unlicensed or licensed bands. Unlicensed bands operate as a public domain and as a first come first serve basis. Examples of systems operating in the unlicensed band include PANs and WLANs. Licensed bands operate under Federal Communications Commission, and in order to use a system on a licensed band, an individual has to pay a service fee. Examples of systems operating in the licensed band include WANs and LANs.

Currently there is a wide variety of devices on the market that enable wireless communication between two points within these systems. Various devices enabling communication within a wireless system include wireless phones, cellular phones, computers, pagers, and blackberry's.

3.3 Motion Detection

There are two basic types of sensors: active and passive. Active sensors include photosensors, automatic door openers, and ultrasonic sound waves. Photosensors send a beam of light in a range. When someone or something breaks that beam of light an alarm or other output signal is emitted. Automatic door openers and ultrasonic sound waves are forms of radar detection. They send out a signal in waves and wait for a reflection of the waves. The time it takes to reflect back determines its distance from the sensor. Passive sensors are those such as infrared sensors. An example of an infrared sensor is a motion detecting light. Motion detecting systems work by emitting infrared light and detecting a sudden change in infrared energy; such as temperature. These sensors can be implemented to sense the temperature of the human body, 93 degrees Fahrenheit, at a wavelength of 8 to 12 micrometers. Infrared light bumps electrons off a substrate which is detected and amplified into a signal [8].

In order to be able to detect from greater distances, these sensors need a Fresnel Lens, similar to a magnifying glass. Most sensors can only detect a few feet without one of these lenses. A Fresnel Lens is a Plano Convex lens collapsed on itself to create a flat piece of plastic. Figure 6 [9] shows a Plano Convex lens on the right and an equaled power Fresnel Lens on the

left. When the Plano Convex lens is collapsed into a Fresnel Lens, the optical characteristics are retained. The ability to make the thickness of the Plano Convex lens much smaller into the Fresnel Lens allows the absorption losses to be much less [9].

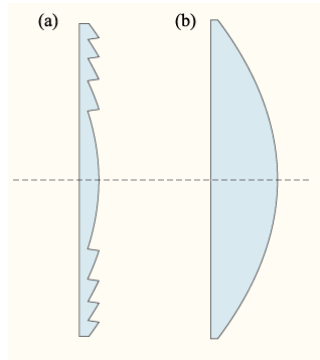


Figure 6: Fresnel Lens and Plano Convex lens

3.3.1 Pyroelectric Sensors

Pyroelectric sensors are passive sensors that emit two infrared energy waves. The purpose of this is to cancel signals caused by vibration, the temperature within the sensor, and sunlight. The sensor will respond to a moving body only, whether it be an animal or human being. It will not detect a stationary infrared source. Pyroelectric sensors are sensitive to changes in air density and can detect any moving body without a Fresnel Lens around 3 feet. However, with a Fresnel Lens detection is substantially increased to 90 feet [9].

3.4 Radio Frequency Identification (RFID)

Radio Frequency Identification (RFID) uses radio frequency to communicate between a reader and transponder. RFID is primarily used for identification, location, and tracking of people, animals, or products. In order for RFID to work, there needs to be a transponder device and a receiver device. The transponder will transmit a signal to the receiver to identify what the item or thing is that the transponder is attached. The receiver takes in the signal from the transponder and then must decode or modulate the transmitted signal to determine the transponders tag. After the receiver modulates the signal, it must then be able to communicate to a computer to take the modulated signal and associate the tag to a name or number.

There are two types of RFID: active and passive. There are also two different ways to design a transponder in RFID: electric coupled transponder systems and magnetic coupled transponder systems.

3.4.1 Active RFID

An active RFID device uses a battery to power its circuitry, thus the transponders reading distance is not limited by the strength of the signal it receives. The active RFID transponder can communicate over 100 meters to a receiver. They are able to collect thousands of tags from a single receiver. The receiver has the ability to detect an active transponder going over 100mph. Active systems are able to monitor and record the sensor status continuously which can be helpful when tracking products across the country. They can apply an accurate time and date stamp every time the receiver flags the transponder. Since active RFID is powered, larger data spaces can be accessed and searched enabling it to transmit longer data packets. They are able to have a read and write storage of roughly 128kbytes which is a dynamically searchable data storage unit.

Active RFID is best suited where business processes are dynamic, the movement of the cargo is variable, and where more complexity of security, sensing, or data storage is needed. Active systems are beneficial because it takes a very low power signal to be transmitted from the receiver and can generate a high-level signal back to the receiver allowing it to reach 100 plus meters [10].

3.4.2 Passive RFID

The transponder in a passive RFID system is energized by a radio frequency (RF) wave transmitted by the receiver. This wave is a time-varying electromagnetic sine wave. As soon as the transponder gets the energy from the wave it immediately transmits a signal back to the receiver with the information stored on the transponder, termed as backscattering. Passive RFID either reflects the energy from the receiver or it absorbs and stores the energy from the receiver to generate a quick response. In order for this to work the receiver must be able to provide a strong signal and must also be able to receive low signal strengths.

There are some essential steps in order for the transponder and receiver to communicate with one another. In the first step, the receiver must continuously generate a radio frequency (RF) sine wave. The receiver will watch for modulation. When the receiver detects modulation it is an indication of a transponder. The transponder must use the RF sine wave energy to power its circuitry. Once the transponder obtains enough energy, it divides the RF sine wave and begins clocking data to an output transistor. This transistor is connected across the coil inputs on typical systems. The output transistor shunts the coil which corresponds to the data which will be clocked out of the transponder's memory. The transistor in parallel with the coil causes fluctuation of the sine wave reflected back to the receiver. This fluctuation is seen as a change in amplitude. The final step in the communication is that the receiver must peak-detect the amplitude-modulated data, then it will process the bit stream received from the transponder.

The idea of the transponder transmitting the data signal to the receiver is known as backscatter modulation. The receiver has to peak-detect the data from the transponder at approximately 60dB down, meaning around 100mV down on a 100V sine wave. The data bits must be encoded to be sent to the receiver. The purpose of data encoding is to alter the data bit stream in-between the time the transponder receives the signal and the transmission is sent back to the receiver.

There are three types of data encoding: Non-Return to Zero (NRZ), Differential Biphase, and Biphase_L (Manchester). NRZ is the most direct approach. The basic idea behind NRZ is that the 1's and 0's are sent directly to the output transistor which is then in turn transmitted to the receiver. Differential Biphase data encoding involves making a transition on every clock edge. This is used when the receiver and bit stream need to be synced, which is done by embedding clocking information. One convenient aspect of Differential Biphase is that it has error correction capabilities meaning the receiver will not receive a piece of data that is not wanted. Biphase_L type is a variation of biphase. The main idea behind the Biphase_L type is that there is not always a transition on the clock edge.

Once the receiver receives the signal from the transponder, it must modulate the data. Along with the three types of data encoding, there are three types of data modulation. The first type is direct. Direct data modulation means that a high is a '1' and a low is a '0'. Direct data modulation has a high data rate; however it has low noise immunity. The second type is frequency shift keying (FSK). This type uses two frequencies for data transfer. A '0' is denoted

by an amplitude modulated clock cycle with a period of the carrier frequency divided by 8. A '1' is denoted by an amplitude modulated clock cycle with a period of the carrier frequency divided by 10. FSK allows for a simple reader design and has strong noise immunity, but has a lower data rate. The third type is phase shift keying (PSK). In this type only one frequency is used. The shift between '1' and '0' is accomplished by shifting the phase of the backscattering clock by 180°. There are two types of PSK modulation. The first type is changing the phase at any '0', while the second type alters the phase at any data change. PSK modulation has average noise immunity performance, a somewhat simple receiver design, and a faster data rate than FSK [11].

Passive systems are limited in communication range by two factors. The first is the need for a very strong signal to power the transponder. Strong signals are able to propagate in a short distance. The other problem is that there is a small amount of power available for the transponders. Most passive devices are only able to be power at 3 meters or less although there have been cases where they can work up to 10 meters. This disadvantage of the passive system makes it appropriate when the movements of the transponders are consistent and controlled.

Passive RFID transponders are hard to identify when there are many in the area. Not only is it difficult to collect many tags, it produces unreliable data. To distinguish between each transponder requires a considerable amount of communication between the receiver and transponder systems. It is usually a multi-step process that consumes quite a bit of time and thus is not very efficient. Along with the time and efficiency, the potential of interference increases with the number of tags trying to communicate with the receiver.

Passive RFID is unable to continuously monitor the status of products and are only able to report the current status of the device. This makes it impossible to track cargo over a great distance at checkpoints. Passive systems also have a small amount of read and write data storage. Usually there is about 128 bytes or less of data storage. Typically there are no search capabilities or data manipulation. The passive RFID systems have the advantage of being relatively cheap. The ability of Passive RFID to be inexpensive and battery less makes it a desirable transponder for less secure applications [10].

3.4.3 Design of a Transponder System

There are two basic design techniques used to design transponders: electric coupled transponder systems and magnetic coupled transponder systems. They both have their pros and cons. Electric coupled transponder systems are able to reach much higher frequencies than the magnetic coupled transponders. Electric coupled systems can be designed for both passive and active whereas the magnetic coupled systems are mostly used for passive RFID. Magnetic coupled systems are able to be designed much smaller than the electric coupled systems due to the lower frequency and amount of power required to operate the circuitry. The next two sections will describe each type in more detail.

3.4.3.1 Magnetic Coupled Transponder Systems

Magnetic coupled transponder systems can operate as high as 29MHz and need protocols for identifying many transponders at once. There are two types of magnetic coupled transponder systems; the 125kHz operating transponders and the 13.56MHz operating transponders. Both types use relatively low frequencies compared to the electric coupled transponder systems.

The 125kHz operating transponder is the most common transponder in today's world. The antenna for receiving and transmitting is comprised of many turns of wire and thus has a very low range of only a few inches. The operating distance for a 125kHz designed transponder system is approximately 2cm. One of the main issues with the 125kHz transponder is that the frequency is low making it difficult to design filters to separate, transmit and receive signals. Another problem is the energizing field is much stronger than the return data field strength. These issues could lead to problems when designing.

The 13.56MHz is also a common transponder design. For these transponders the antenna does not need as many turns as the 125kHz transponders. They can be made extremely small at approximately 1.5cm x 1.5cm (0.6" x 0.6") in area. The transponder systems are able to extract the energy needed for the system and data from the reader and then communicate back to the reader. The 13.56MHz systems have both read and write capabilities and also have anti-collision properties. The typical range of a 13.56MHz system is 50cm-1m [12].

3.4.3.2 Electric Coupled Transponder Systems

The advantage of using electric coupled transponder systems over magnetic coupled transponders is the increase in ranges the transponder can transmit across. Electric fields radiate out from an energizing antenna which requires the antenna systems to be half the wavelength of the operating frequency. As an example, for 100MHz a 150cm (59.1”) antenna is needed whereas a 5.8GHz signal needs a 2.5cm (1”) antenna. The issue in designing with high frequencies is that they are more expensive and lose the ability to “transfer energy at the rate of the inverse of the wavelength squared” [13]. For example, a 2.45GHz needs seven times the amount of energizing fields of a 915MHz signal. For passive transponders, the range is limited from 10 to 15 meters. Electric coupled transponders are as inexpensive as 15 cents because of the ability to make them into small integrated circuits. The 15 cent transponder is passive, has no onboard tuned circuits, is read only, consists of a single integrated circuit (IC), a simple antenna, operates at any frequency, is temperature insensitive, and broadcasts large data values. The receiver to electric coupled transponders can be complex because the receiver needs to provide frequency stability, energy to the transponders, and receiver selectivity. Electric coupled transponder systems are ideal for situations where there is one receiver with many tags [13].

3.5 E-ZPass

Given our designs application, considering alternative systems that are currently on the market and how they operate gave us insight as to what technology may best suit our needs. One comparable device on the market is the toll paying system E-ZPass. This system gained much attention in 2002 in Massachusetts, and even earlier in some more southern states.

E-ZPass is an RFID based system for paying traffic tolls without having to stop to deposit change at a toll booth. This system has the ability to recognize individual transponder signals, and use that data to perform an action with that particular transponder signal. This action is completed through use of a transponder within the vehicle and a receiver in the toll complex. In essence, the receiver recognizes that a transponder has come into range, and communicates with it to obtain the transponder’s stored data. Once the data has been read from the transponder, the system looks up the account associated with the particular transponder, and performs the subtraction of the toll from the account.

Being able to uniquely identify different tags and associate those identifications with stored information is the essence of this project.

3.6 System Analysis

These various options present several efficient ways to provide containment security for individuals in nursing homes who are at risk of wandering beyond the facility doors. The system needs to detect someone attempting to stray into an area where they are not allowed. The system also needs to be able to identify the individual and the doorway which they are trying to wander through.

The following table was used to compare the requirements versus each system architecture discussed. Within the table each system was allocated a score from 0-4 (0 being poor, 4 being excellent) on how well it matched our requirements. The scores were multiplied by the weights given next to the requirements column and the columns were added with the totals in the bottom row showing us which design best matches our needs; the one with the highest score.

Table 1: System Analysis
 (0=Poor, 1= Fair, 2= Average, 3= Above Average, 4= Excellent)
 (Weight: 2 – 5, lowest – highest priority)

	<i>Weight</i>	GPS	Magnetic Induction	Motion Detection	RFID (Active)	RFID (Passive)
Functionality	5	0	2	0	0	4
Lifetime	4	3	3	3	3	4
Durability	4	4	3	2	3	3
Cost	3	0	2	2	3	3
Weight	2	2	1	3	3	4
Size	2	2	3	3	4	4
Total		11	14	13	16	22

A Global Positioning System (GPS) system would use triangulation from satellites to determine the location of a person wearing a GPS locator to determine if they were outside a specific perimeter. However, this system would be overly complex for the task at hand, because

we do not require position data of every patient at risk of wandering. There were also the additional considerations of the accuracy of GPS (especially indoors), the cost of a GPS tracking system, and the ethics of tracking a person's whereabouts at all times. These considerations have led us to decide that GPS is too complex a system to be effectively used for our purposes.

Magnetic induction would provide coverage of the doorway, but lacks the ability to distinguish between individuals. It does however possess the ability to distinguish between staff members and residents, because it would only produce a response when someone wearing a special coil passed through the magnetic field. This would be achieved by giving patients at risk of wandering a special coil that would trigger the system. However, this system still lacks the ability to identify individual coils passing through the field, which is a key component of the project.

Motion detection also would provide blanket coverage of the area we wished to monitor (in our case, the doorways), but lacks the ability to identify a specific individual or distinguish between staff members who would be allowed through the door and residents. Therefore, motion detection would not be adequate for the needs in this project.

RFID uses radio waves to transmit data between transponders and antennas. Each transponder is encoded with a particular identification number, and able to be read by the antenna. With an antenna near the door, we would be able to monitor the door, and uniquely identify each transponder to pass into the range of the antenna. If we use an active system, the receiver has the potential to detect anyone within 100 meters. If we use a passive system for a transponder, we can make the item being identified very small and lightweight since no battery is required. Since a passive system would not need batteries the lifespan of the system would be extended as opposed to an active system which requires a battery. This is our initial choice for a system to use when providing an alert notification system for people likely to wander away from nursing homes. Specifications on the design will be discussed in the following sections.

4.0 Initial Designs

With the requirements determined and a general system approach defined based on how well it matched our requirements, a system design was developed.

4.1 Design I

Our original design consisted of a 134kHz 22mm passive transponder (supplied by Texas Instruments), a transceiver, a receiver, and antennas for communication between the devices. The transponder would be housed in a small casing device that would be worn by the individual who required monitoring. Once the individual were to walk through an exit door equipped with an antenna to pick up the transponders signal, it would then activate the reader. The reader would acknowledge the signal and read the identification number relating to the transponder uniquely identifying the individual to whom the transponder was assigned.

4.1.1 Block Diagram

Before designing the individual components to our original design, we developed a block diagram to ensure that we had taken all parts into consideration in regards to what components will complete which functions and how they will interact with one another.

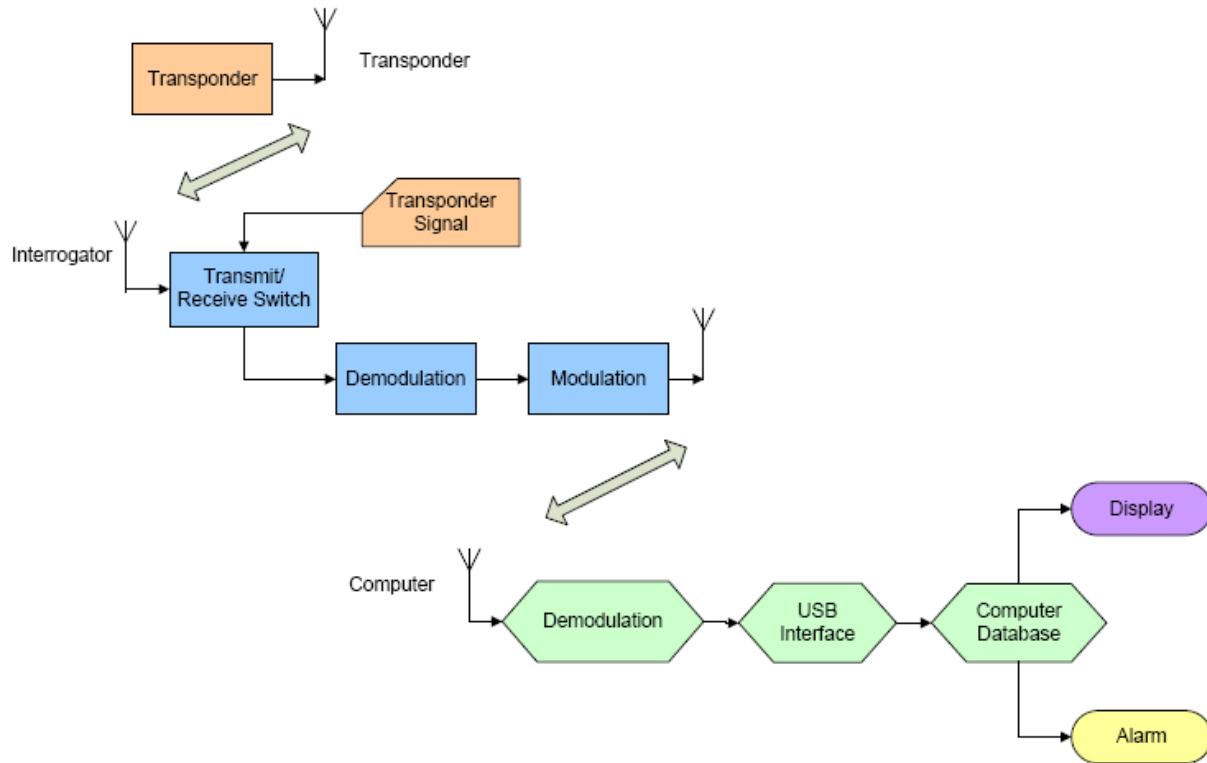


Figure 7: Design I - Block Diagram

The transponder being worn by the individual would emit a signal that would be picked up by an antenna located at an exit door. This signal would then be sent to the receiver (i.e. interrogator) also located at the door via another antenna. The receiver would take in the signal which it would have to demodulate. Once the receiver had identified the signal, it would then modulate the signal and send it via another antenna to the computer. As the computer received the signal it would have to demodulate it and would send it to the USB interface. From the USB interface the demodulated signal would go to the computer database where it would set off the alarm (flashing lights) and display the identified individual.

4.1.2 Analysis

With this design, we found ourselves to be pursuing a direction in which the issues we were encountering with the individual components just described were proving to be more time consuming than anticipated. Originally designing our system from scratch created many issues in our testing.

4.1.2.1 Design Complications

Design I had a number of issues that made it an inefficient process to accomplish our task. One problem we encountered came from our antennas being a piece of wire and our power supply only being capable of outputting a 5Vp-p signal. This meant that we were unable to transmit enough power to the transponder to allow it to transmit its data back to the receiver. A basic RC low and high pass filter was used to remove excess noise into the FSK demodulator, however the filter roll off was too shallow to have a positive impact on the signal to noise ratio. This decision prompted our choice of a Sallen Key filter block.

Testing of the FSK demodulation IC revealed that with the recommended configuration from the data sheet required a frequency division of 8 or 10 to produce any outputs when configured for the 129.7kHz center frequency; something that was not mentioned in the data sheet. Another problem encountered was creating a -5V rail which the filter and amplification stages required to function properly. Without the negative rail, the filter stage would remove the negative half of the waveform. There was also difficulty in finding direction for the creation of a USB driver to provide an interface between our receiver at the computer end and the actual computer.

Given the various complications in attempting to get the device to work, no test results or other analysis were able to be performed. Without the system running, testing was unable to be done.

4.2 Design II

Given the various complications we were having with Design I, redefining a new method to design our product was necessary in order to complete our system for our projected due date. As goes with many products, there are various ways to achieve the same end goal; all with their different pros and cons.

For our purposes we began to consider alternatives to designing our components from scratch and began to look into purchasing various components that when integrated would achieve our same end goal.

Given that Texas Instruments (TI) offers complimentary samples of components to entice users to use their products for future projects, our first direction when looking into RFID systems

were those created by TI. However, after further inquiry it was found that TI does not provide complimentary samples or University discounts for students on their RFID systems. With this, we did not limit ourselves to which type of products we were able to choose from due to their stipulations on what can and cannot be offered as samples or at reduced costs.

While performing further research on RFID systems that would be feasible to implement for our design, using various frequencies was also considered. As a basis for our designs in each frequency range, we used our original block diagram which told us we essentially need a transponder, receiver, interface, and antennas. At least two antennas would be used in any case in order to account for the various orientations of the transponder or location of the transponder on the individual to ensure the signal would be read. After speaking with technician specialists from both TI and various vendor companies, pros and cons were found within each frequency range we could have pursued. A value analysis performed at the end of the options described after the following sections helped to narrow down an option.

4.2.1 Low Frequency (LF) – 134kHz

Staying with our original design consideration and pursuing a product that uses a passive transponder in the low frequency range of 134kHz would be beneficial as the transponder can be read through the body. This would alleviate the concern of the body shielding the transponder and blocking the antenna signal. However, passive transponders used in low frequency do not have anti-collision properties. This would cause issues if two individuals with transponders were to try to exit the door; potentially only one or neither signal would be received. Also, with a passive transponder, the orientation of the transponder reflects greatly on the antenna being able to receive its signal. Another concern relating to using a passive transponder in the low frequency range is that its signal range is considerably lower than that of a transponder used in other frequency ranges or that of an active transponder.

A low frequency system option was developed with the possible pros and cons kept in mind. This system was designed as an option that would later be compared to a high frequency and ultra high frequency option to determine which frequency option would be most feasible.

For the low frequency option, a 120mm transponder would be used with a Series 2000 High Performance RFM, a medium gate antenna, and a Series 2000 Standard Reader.¹

LF – Option 1



Figure 8: Low Frequency Design Option

The 120mm transponder would be more ideal than the 22mm transponder which we currently have given that it has a larger internal antenna and can transmit signals for a greater distance than that of the 22mm transponder. The rest of the components are complimentary components of one another chosen based on our desired reading range and operating conditions. For instance, we choose a medium gate antenna as opposed to a stick antenna that would not be capable of reading the signal from a passive transponder for the distance required. We also did not follow the path of choosing the large gate antenna due to its size (40" x 20.5" x 2") considering we were looking to place the antenna discretely by the doorway.

¹ Information regarding system pricing and images were obtained from [14].

LF – Option 1

• 120mm Transponder (P-7532)	\$14.32
– Higher mm winding = greater read range	
• Series 2000 High Performance RFM (P-7547)	\$290.36
• Medium gate antenna (P-7561)	\$255.00*2
• Series 2000 Standard Reader (P-7539)	\$705.84

Total = \$1,520.52

Figure 9: Low Frequency Design Costs

The various code numbers next the components are the id numbers for the products via the Dynasys website (www.rfidusa.com). The costs as seen are a reflection of how much each part would cost when ordered through Dynasys.

4.2.2 High Frequency (HF) - 13.54MHz

One concern with our product is that we need to ensure that all patients wearing transponder tags are acknowledged when they go through an exit door and the system alarms. A system that does not offer anti-collision jeopardizes our products functionality. However, a system that works in the high frequency range would be able to handle anti-collision situations.

A high frequency system option would include encapsulated transponder tags, a Feig Mid Range Reader and USB, and a Series 6000 Gate Antenna.²

² Information regarding system pricing and images were obtained from [14].

HF – Option 2



Figure 10: High Frequency Design Option

The Series 6000 Gate Antenna was chosen as opposed to its Large Antenna counterpart option due to the convenience of size. The reader of this system would be considerably small (the size of an average pen) and would fit discretely by the door connected to the antenna awaiting the signal from our hefty durable encapsulated transponder tags. These tags would prove to be durable enough to withstand a drop from 6' onto a concrete surface as described in our requirements section without the added need of a special case.

HF – Option 2

- | | |
|------------------------------------------------|------------|
| • Encapsulated Transponder (P-10401) | \$3.66 |
| – Heavily durable | |
| • Feig Mid Range Reader (USB) (P-9870) | \$538.00 |
| – Dependent on antenna, up to 40cm range (15") | |
| • Series 6000 Gate Antenna (P-8239) | \$488.75*2 |
| – Recommended by Dynasys | |

Total = \$1,519.16

Figure 11: High Frequency Design Costs

Taking costs into consideration, both the low frequency and high frequency options are nearly equal in that respect. However, our value analysis will provide better insight as to an ideal choice given our requirements.

4.2.3 Ultra High Frequency (UHF) - 900MHz

Our final option for the various frequency ranges is that of ultra high frequency. When working with RFID systems in the ultra high frequency range, we found that similar to that of high frequency devices, it is capable of anti-collision. However, in ultra high frequency the transponder tags are unable to be read through the body. This means that an individual could potentially shield the transponder with their body were not enough antennas placed around the doorway.

The system we found that would be most applicable to our project in that of the ultra high frequency range was to implement a business kit provided by Dynansys.

This business kit comes complete with a variety of tags to experiment with, an enclosed reader, a circular polarized patch antenna, interface board, serial cable, and software to guide a novice through the programming procedures.³

³ Information regarding system pricing and images were obtained from [14].

UHF – Option 3



Figure 12: Ultra High Frequency Design Option

However, one problem with purchasing the kit is that it can only be used with the antenna provided, and no additional antennas. A circular polarized antenna would mean that the orientation of the tag does not need to be taken into consideration as much as it would have to be if the antenna were a linear polarized antenna. Nevertheless, there lies a possibility that the individual's body could block the antenna from picking up the signal from the transponder were the antenna not oriented properly.

On a positive note, if we were not satisfied with the variety of tags supplied in the kit we could purchase a tag of our choice provided it is applicable for the ultra high frequency range. An example of a tag we would find ideal to include with this option is depicted in Figure 12. The EPC Mini Metal Tag would be inexpensive enough to purchase for several individuals and is small enough to incorporate in a small casing design for the individual to wear.

UHF – Option 3

- UHF Gen 2 Reader Antenna Business Kit (P-10412) \$1,295.00

Total = \$1,295.00

Figure 13: Ultra High Frequency Design Cost

4.2.4 Value Analysis

After analytically comparing the various frequency range choices, a value analysis as a numerical representation of how well each system fulfills our requirements was performed. In our first analysis, we compared each systems capability in regards to anti-collision, interfacing, and cost. The rating system used was as follows:

Anti-Collision

- 0 - Does not have Anti-Collision
- 1 - Does have Anti-Collision

Interfacing

- 1 - Poor (No software, no easy interface with PC)
- 2 - Fair (Difficult interface implementation)
- 3 - Good (Software and interface with PC, little documentation)
- 4 - Excellent (Easy to install and apply)

Cost

- 1 - High Cost (over \$2000)
- 2 - Medium Cost (\$1500 - \$2000)
- 3 - Medium-Low Cost (\$1000-\$1500)
- 4 - Low Cost (Under \$1000)

These values are allocated to the various designs in a value analysis in the following table. Each value was multiplied by the weight associated with our requirements with 0.8 being of high priority and 0.3 being of lower priority. These values were then added together for each device in the subtotal row; the highest number being our most ideal solution.

Table 2: Design II - System Value Analysis

Requirements	Weights	LF (134.2kHz)	HF (13.54MHz)	UHF (900MHz)
Anti-Collision	<i>0.8</i>	0	1	1
Interfacing (software, USB, etc.)	<i>0.5</i>	2	3	4
Cost	<i>0.3</i>	2	2	3
Subtotal		1.6	2.9	3.7

For the next analysis, we compared each system's reading range ability and the various antenna size options. The rating system used was as follows:

Reading Range

- 1 - Very poor range (Less than 5'')
- 2 - Low Range (6'' – 12'')
- 3 - Medium Range (13'' – 24'')
- 4 - Large Range (24'' – 10')
- 5 - Largest Range (over 10')

Antenna Size

- 5 - Less than 6'' X 6'' x 1''
- 4 - ~ 8'' x 8'' x 1''
- 3 - ~ 24'' x 12'' x 1
- 2 - ~ 40'' x 20'' x 2
- 1 - Larger than 60'' x 30'' x 2''

Table 3: Design II - Antenna Value Analysis

	Weights	LF (134.2kHz)				HF (13.54MHz)			UHF (900MHz)	
Antenna Type		<i>Stick</i>	<i>Sm gate</i>	<i>Md gate</i>	<i>Lg gate</i>	<i>Sm gate</i>	<i>Md gate</i>	<i>Lg gate</i>	<i>Yagi</i>	<i>Patch</i>
Reading Range	<i>1</i>	2	3	3	3	2	3	3	5	4
Antenna Size	<i>0.65</i>	5	4	3	2	3	2	1	1	4
Subtotal		5.25	5.6	4.95	4.3	3.95	4.3	3.65	5.65	6.6

After preparing an analysis of the various antennas we could choose from each frequency range, an analysis combining the two comparisons is in the following table.

Table 4: Design II - Value Analysis

	LF (134.2kHz)				HF (13.54MHz)			UHF (900MHz)	
Antenna	Stick	Sm gate	Md gate	Lg gate	Sm gate	Md gate	Lg gate	Yagi	Patch
Total	6.85	7.2	6.55	5.9	6.85	7.2	6.55	9.35	10.3

From this overall value analysis we can now determine not only which system is most ideal for our purposes, but also which antenna to choose. From the numbers given it would appear that the best option to pursue would be within the 900MHz range of UHF with the patch antenna.

4.2.4.1 Design Complications

Given our value analysis, it would appear that designing our product in the UHF range with the patch antenna would be most ideal. However, without the use of complimentary samples or University discounts, the costs of various RFID systems proved to be considerably expensive. In addition to the cost dramatically exceeding our budget, none of the devices would have been able to fulfill all of our requirements without considerable drawbacks in one or more of the following areas: cost, size, or reader range. Given that the size and reader range contribute dramatically to our functionality, Design II was deemed unfeasible and no system was purchased or tested. With this conclusion, several more detailed requirements were defined, and an alternative direction was sought out.

5.0 System Design Requirements

After pursuing two design directions, careful consideration as to why the designs failed was analyzed resulting in new design requirements. The following requirements allowed decisions based from numbers calculated to ensure the product would work as designed rather than choosing what components seem to be applicable.

In pursuing this direction, a few things that were taken into consideration in order to follow our requirements as originally stated was ensuring that if two transponder signals were sent to the receiver in unison – that neither data would be lost, the security of the information transmitted wirelessly, battery life, and reading range.

5.1 Aloha Transmission Encoding

Aloha transmission systems come in two popular versions, un-slotted (pure) and slotted. Pure systems generally comprise multiple stations that are both transmitting data and receiving it. These stations attempt to transmit their data whenever it is ready to be transmitted. For example, station 1 is trying to transmit a packet of data to station 2. During the time it takes for station 1 to attempt its transmission, station 2 has data that needs to be sent to station 1. Both stations end up trying to transmit data to each other at the same time, and the packets collide effectively destroying both packets of data. This transmission system is not very efficient, because if there is any degree of overlap, both overlapping packets are rendered useless.

Slotted Aloha is a variant of pure aloha where an average packet length is determined and agreed upon by all the stations. The stations are synchronized to understand that there are particular windows of the average packet length available for the transmission of data. Therefore, when a station has a packet of data ready to transmit, it waits for the start of the next slot to attempt its transmission. This eliminates the partial overlap which makes pure aloha inefficient. The packets either overlap completely, or not at all. Once it is detected that there has been a packet collision, both stations wait a random interval before attempting to transmit their data again. This also reduces the probability of a packet collision [15].

5.2 Encoder/Decoder Pairs

Since our project involves associating transmitted data with personal information about a patient and transmitting it wirelessly, our device requires transmission security. Even though we are not associating an ID number (or other manner of patient identification) with the files that might be stored in the nursing staff's computers, we are associating it with their names. That alone is secure enough information that should not fall into the wrong hands.

In order to provide wireless security, encoders and decoders were taken into consideration. The pair we felt best accomplished this was the Holtek 2¹² Encoder/ Decoder pair because it has the potential to offer 2¹² different addresses to which data could be sent. If the addressing on the decoder does not exactly match the addressing on the encoder, there is no data output from the decoder [16] [17].

This is a rudimentary level of information security, so for a final iteration, the system could use the wireless network standards of security by allowing our device to hop onto an existing network inside the nursing facility for maximum information security.

5.3 Power Analysis

The transponder for our project requires a total power consumption of 8.75mA per hour. When the PIC18F252 is in RCIO mode, it will consume 0.75mA per hour to operate, and the transmitter TLP-434A will consume 8mA per hour. To provide this amount of current at a TTL logic level for approximately one year, a power supply capable of supplying current for 8,760 hours is needed.

$$\frac{1 \text{ year}}{1} \left| \frac{365 \text{ days}}{1 \text{ year}} \right| \left| \frac{24 \text{ hours}}{1 \text{ day}} \right| = 8760 \text{ hours} \quad (5.1)$$

Another aspect that needs to be considered is that a typical battery voltage is 1.2V for rechargeable batteries, and 1.5V for alkaline batteries. Our project would utilize two batteries connected in series to provide a logic level of either 2.4V or 3V. Therefore, a transition from voltage levels under TTL levels up to TTL levels is required. One option that was considered for this project is a boost converter. The efficiency of the chosen boost converter, the UCC3941-5 is approximately 85% given the parameters of an input voltage between 2.4V and 3V, and an output current of 8.75mA.

The total current drawn from the batteries can be determined through the calculation of:

$$\frac{8.75mA}{0.85 \text{ efficient}} = 10.29mA \quad (5.2)$$

With this knowledge, to attain the goal of having the transponder capable of operation for 1 year, a battery with 90,140.4mAh would be required.

$$10.29mA * \frac{8760 \text{ hours}}{1 \text{ year}} = \frac{90140.4mAh}{\text{year}} \quad (5.3)$$

Based on this calculation, it was determined that utilizing rechargeable batteries that would last for approximately a month at a time would be a more effective option. These rechargeable batteries would need to be capable of producing 3,511mAh per battery if two are assumed to be utilized in the final design.

$$\frac{90140.4mAh}{\text{year}} * \frac{1 \text{ year}}{12 \text{ mont hs}} = \frac{7511.7mAh}{\text{mont h}} * \frac{\text{total}}{2 \text{ batteries}} = \frac{3755.85 \text{ mAh}}{\text{mont h} * \text{battery}} \quad (5.4)$$

If size constraints limit the number of batteries to one, then the single battery would be required to supply 7,511.7mAh/month. A typical rechargeable AAA battery can only supply 850mAh, and a typical rechargeable AA battery can supply 2,000mAh. Batteries that were considered at include:

Table 5: Battery Considerations

Battery Type (rechargeable)	Current	Voltage (V)	Cost (\$)	# of batteries	size	Cost per battery (\$)
Duracell	1800mAh	1.2	14.99	4	AA	3.7475
Energizer	2500mAh	1.2	24.99	8	AA	3.12375
Energizer e2 Lithium	2500mAh	1.2	18.99	4	AA	4.7475
Tenergy	2600mAh	1.2	1.69	1	AA	1.69
Camelion	3500mAh	1.2	12.98	2	C	6.49

However, if it were possible to include multiple batteries in the transmitter, the need for current could be satisfied using three of the Tenergy or Energizer batteries in series, or two of the Camelion batteries in series. In the interests of saving space the best choice is to utilize the three AA batteries, but the battery types must not be mixed in the device (for example, using one Energizer, one Energizer e2, and one Tenergy) because different battery chemistries may produce hazardous chemical reactions.

The most feasible candidate for use in the project is the Tenergy 2,600mAh rechargeable battery. This is based on the table below. A weighting was assigned to three categories: current, size, and cost per battery.

Category	Weight	Duracell	Energizer	Energizer e2	Tenergy	Camelion
Current	5	3	4	4	4	5
Size	5	4	4	4	4	2
Cost per Battery	2	3	3	2	5	1
Totals (sum of weight * value)		41	46	44	50	37

5.4 Reading Range

The range required to produce detectable and reliable transmissions was determined given a few considerations. The first consideration was the size of the doorways that would need to be covered by the SaunterNot system. According to the Americans with Disabilities Act, 32 inches is the minimum allowable door width measuring from one edge of the doorframe to the opposite doorframe. However, it is also mentioned that in acute care hospitals bedroom doors are at least 44 inches wide.

With 44 inches being the worst case scenario for a possible door width, it was also inferred that a double door would be at maximum 88 inches wide. Therefore, the minimum detectable range to ensure coverage of the entire door width is 88 inches. Another factor to consider is the need to blanket the entire height of the door. Assuming that the doorway mounted receiver will be mounted above the door, it is assumed that the door height is 8 feet as a worst case estimate. Therefore the minimum range that needs to be covered by the doorway mounted receiver is 96 inches [18].

On average, a healthy woman walks at a speed of 3 miles an hour. Males have a tendency to walk more quickly at 3.5 miles an hour [19]. For the assumed worst case scenario of 3.5 miles an hour, the average walking person covers:

$$\frac{3.5 \text{ miles}}{\text{hour}} \times \frac{5280 \text{ feet}}{\text{mile}} \times \frac{1 \text{ hour}}{60 \text{ min}} \times \frac{1 \text{ min}}{60 \text{ seconds}} = \frac{5.13 \text{ feet}}{\text{second}} \quad (5.5)$$

If it is assumed that the minimum range extends perpendicularly from the doorway 8 feet, and that our individual is walking at an average speed of 5.13 feet per second, it would take 1.56 seconds for a person to pass completely through the minimum required range of the doorway.

$$\frac{1 \text{ second}}{5.13 \text{ feet}} \times 8 \text{ feet} = 1.56 \text{ seconds} \quad (5.6)$$

6.0 Functional Entities

After various calculations, a more structured set of requirements was synthesized. With the new list of requirements, specific components could be chosen to perform the tasks needed for our third design to operate.

6.1 Block Diagram

With several essential factors already calculated and taken into consideration, moving onto a general block diagram of parts that would be needed for the design was next.

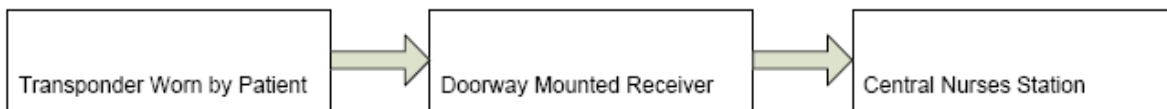


Figure 14: Design III - Block Diagram

After recognizing what different blocks will perform which tasks, designing the individual components based on our calculations is described in the following section.

6.1.1 Transponder

A device is needed that can continuously transmit a unique identification number. This unit also needs to be portable seeing as it is worn by the patients. In order for this to happen the use of an external power source is needed. As mentioned before different battery options were analyzed to give the transponder a lifespan of 1 year. The transponder block consists of a PIC microcontroller, boost converter, and transmitter in order to accomplish these tasks.

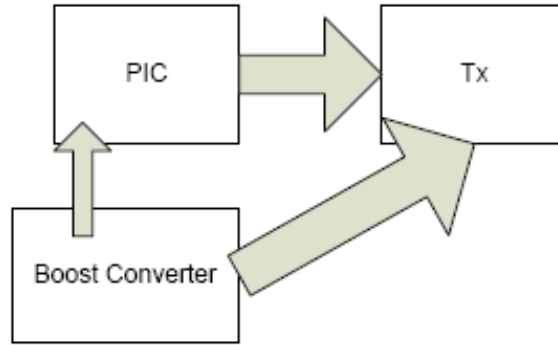


Figure 15: Transponder Block Diagram

6.1.1.1 Boost Converter

In order for the transponder to be used on a patient anywhere, a battery needs to be implemented. The voltage supplied by the batteries was 3V. The PIC microcontroller and transmitter require 5V to be operational. Thus a device to convert 3V to 5V is needed. This device needed to have a high current efficiency in order maximize battery life.

After researching numerous boost converters, the UCC3941N-5 was chosen. This particular boost converter had a minimum input voltage of 1V and a maximum input voltage of 5.5V. This is a desirable feature because when the battery begins to lose its charge, the boost converter is able to maintain the output required of 5V. Another advantageous feature of this boost converter is its efficiency.

The UCC3941N-5 is used to convert the battery voltage to 5V. This battery voltage must be less than the output voltage of 5V. Figure 16 shows the configuration of the boost converter as described in the datasheet.

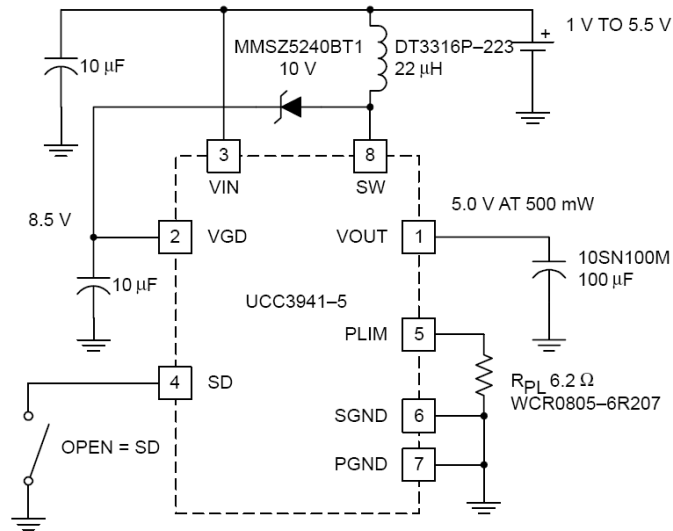


Figure 16: Boost Converter Configuration per DataSheet

In order to be applicable to our application, minor changes have been made from this configuration. One alteration is that pin 4 is connected to ground disabling the shutdown feature. Another slight modification is the power limiting resistor. This resistor was changed to 100Ω. The inductor chosen had to be below 150mΩ ESR and the output capacitor had to have a low ESR. The need for these two components to be lowered was due to the ripple of the output voltage. A lower ESR rating on the inductor and capacitor would produce a lower output voltage ripple. A schematic of the boost converter configured for the devices operational requirements is shown in Figure 17.

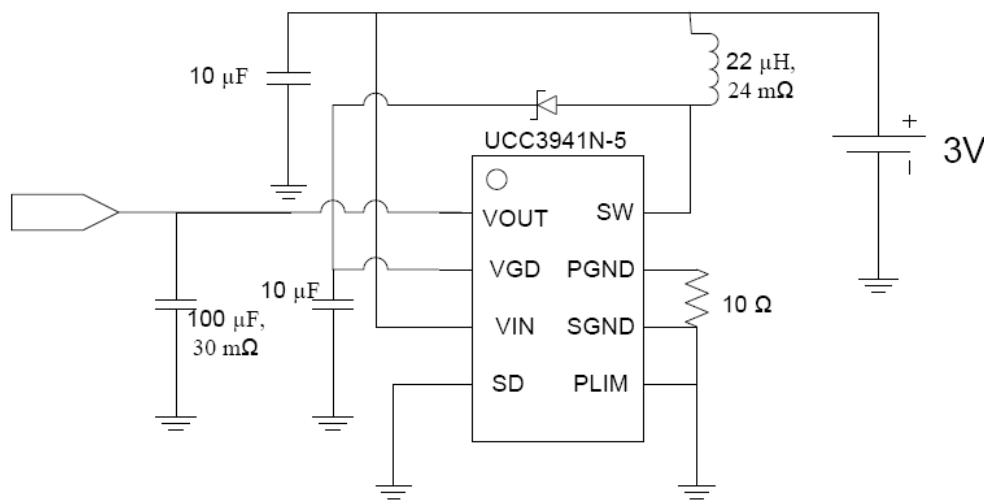


Figure 17: Boost Converter Configuration per Design III

6.1.1.2 PIC

The heart of the transponder lies in the ID number. There needs to be enough available unique ID numbers in order to satisfy a facilities needs. For the purpose of this project an 8-bit binary ID number was chosen. An 8-bit binary number would allow for up to 256 unique identification codes. The 8-bit binary number would need to be stored on a device in such a way so the reader could detect the incoming signal. This device also needs to provide a start sequence to tell the reader when to store the ID number and transmit it to the computer.

There are a variety of devices out there that can potentially complete this task. One possible device is a sequencer. The sequencer takes parallel binary input and produces a serial output. This is done with the use of flip flops and digital logic such as AND gates. Along with the sequencer, a clock and counter would have to be implemented in order for the sequencer to work. First the parallel inputs are loaded into the sequencer based on the counter. When there is a rising edge on the clock, the sequencer will take in a known value; either ground or V_{cc} based on another input pin. It would then shift that into the already loaded data outputting the first value of the sequence, or the LSB. The counter is used to keep track of how many bits are on the input which ensures that only the data loaded in will be shifted out. This system is a possible solution, however for the needs of conserving power and space is not practical for the needs of the transponder.

Another solution would be using a PIC microcontroller. The PIC microcontroller allows for parallel in and serial out; similar to the sequencer, but does so with significantly less power consumption and using only one chip rather than three. The PIC microcontroller can be easily programmed thus making changes to the ID number a simple task. Another desirable feature of the PIC microcontroller are the different oscillator modes available. The mode determines the amount of current drawn and the overall system speed. There are numerous PIC microcontrollers available on the market. Table 6 shows the various considered options for the PIC microcontroller.

Table 6: Options for PIC microcontroller

	Shop Owns	I/O Pins	RAM	Program Memory	Price
PIC16F877	Y	33	368	14K	\$12.70
PIC18F1320	N	16	256	8K	\$5.23
PIC18F252	Y	25	3804	32K	\$7.63
PIC16F84A	Y	13	68	1.75K	\$5.88
PIC18F620	Y	36	3804	64K	\$9.53
PIC18LF452	Y	36	3804	48K	\$8.95

The PIC18F252 was chosen based on the table above. Several reasons for the selection include its availability from the shop making it convenient to obtain and that it had just as much memory as some of the more expensive options, but at a lower cost. Given time constraints and our system requirements, these factors made the PIC18F252 a viable choice for the transponder. After deciding which model to choose, deciding how to program the PIC microcontroller to meet the specifications was the next task

In order to ensure that the receiver will pick up the transponders signal if in range, the transponder will need to continuously transmit its encoded signal. With the PIC18F252, not only will it continuously transmit its encoded signal, but it also has different oscillator modes and multiple programmable functions. The code written for the PIC can be found in Appendix C.

When setting the PIC, the RCIO mode was chosen since the PIC will consume less power than with RC or other possible modes. The internal clock of the PIC was able to be controlled by choosing different capacitances as shown in Figure 18.

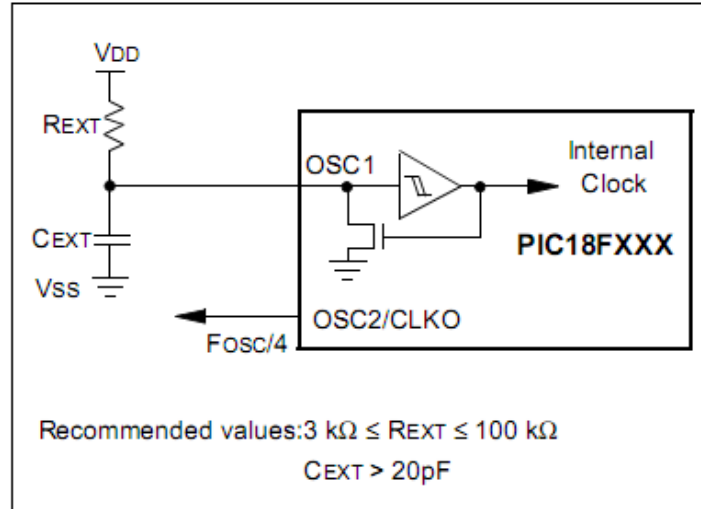


Figure 18: Internal Clock of the PIC

The clock was chosen to be 1MHz. This was chosen because of the desired baud rate and error rate that could be expected from the baud rate as will be mentioned later. Knowing that the frequency desired was 1MHz, the resistor and capacitor could be calculated using:

$$F_{osc} = \frac{1}{R_{EXT} \times C_{EXT}} \quad (6.1)$$

If C_{EXT} is chosen to be 75pF, then the value for R_{EXT} can be computed. The resistor needed for a frequency of 1MHz is:

$$R_{EXT} = \frac{1}{F_{osc} \times C_{EXT}} = \frac{1}{(1 \times 10^6)(75 \times 10^{-12})} = 13.3\text{ k}\Omega \quad (6.2)$$

These values fit within the parameters shown in Figure 18.

When coding the PIC, the TXSTA register of the USART is configured to 10000000b in binary to enable the serial port. The RCSTA was configured so that the serial data flow was enabled. The SPBRG was configured to have 2400bps (25d or 19h). The next portion of the code initiates the USART function of the PIC. The first thing transmitted is the start sequence, 0x7Fh (01111111b). After loading the data into the transmit data registry (TXREG), the code then waited until the TRMT bit was set which designated that the register was empty. The TXREG takes the data in that register, in this case a parallel configuration, and outputs a serial non-return to zero (NRZ) modulation that can be interpreted by the receiver. After the empty bit was noticed the ID number was sent next. The code chosen as an ID number was 0xAEh

(10101110b). The code again waited until the TRMT bit was set which designated that the register was empty. The code then waited a short delay before repeating the process again.

A baud rate of 2.4kbps was chosen. For this baud rate the SPBRG needed to be configured to 0x19h (25.0d). Figure 19 shows the functional block diagram of the USART transmit function.

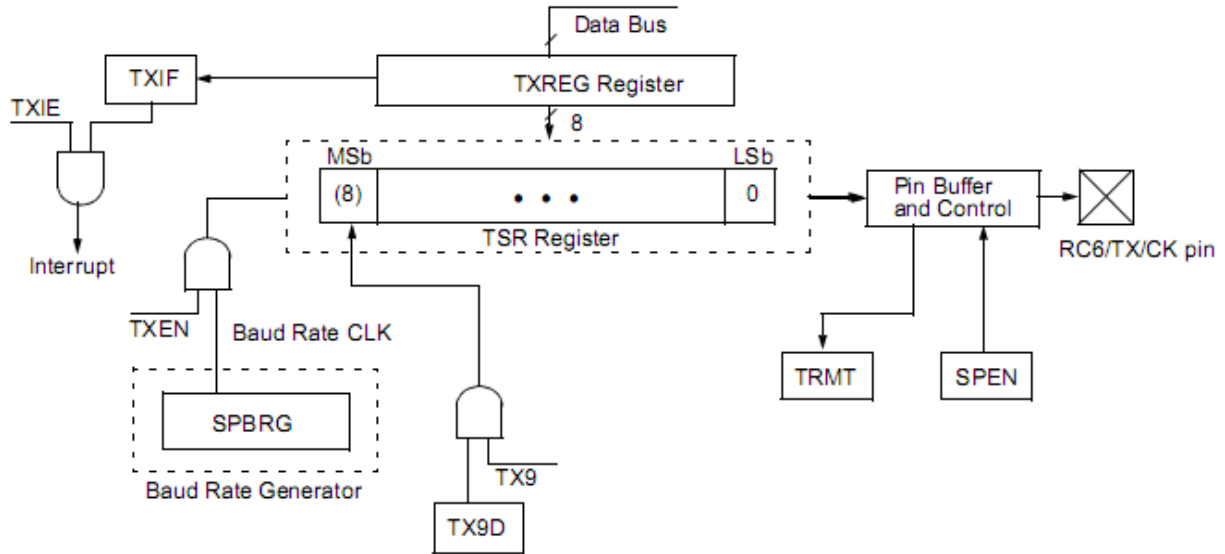


Figure 19: USART Block Diagram

Table 7 shows the different baud rates for various frequencies, their associated error margins, and the decimal value to code.

Table 7: Baud Rates and Error Margins for PIC

BAUD RATE (Kbps)	Fosc = 40 MHz			33 MHz			25 MHz			20 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	NA	-	-	NA	-	-	NA	-	-	NA	-	-
1.2	NA	-	-	NA	-	-	NA	-	-	NA	-	-
2.4	NA	-	-	NA	-	-	NA	-	-	NA	-	-
9.6	NA	-	-	9.60	-0.07	214	9.59	-0.15	162	9.62	+0.16	129
19.2	19.23	+0.16	129	19.28	+0.39	106	19.30	+0.47	80	19.23	+0.16	64
76.8	75.76	-1.36	32	76.39	-0.54	26	78.13	+1.73	19	78.13	+1.73	15
96	96.15	+0.16	25	98.21	+2.31	20	97.66	+1.73	15	96.15	+0.16	12
300	312.50	+4.17	7	294.64	-1.79	6	312.50	+4.17	4	312.50	+4.17	3
500	500	0	4	515.63	+3.13	3	520.83	+4.17	2	416.67	-16.67	2
HIGH	2500	-	0	2062.50	-	0	1562.50	-	0	1250	-	0
LOW	9.77	-	255	8.06	-	255	6.10	-	255	4.88	-	255

BAUD RATE (Kbps)	Fosc = 16 MHz			10 MHz			7.15909 MHz			5.0688 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	NA	-	-	NA	-	-	NA	-	-	NA	-	-
1.2	NA	-	-	NA	-	-	NA	-	-	NA	-	-
2.4	NA	-	-	NA	-	-	2.41	+0.23	185	2.40	0	131
9.6	9.62	+0.16	103	9.62	+0.16	64	9.52	-0.83	46	9.60	0	32
19.2	19.23	+0.16	51	18.94	-1.36	32	19.45	+1.32	22	18.64	-2.94	16
76.8	76.92	+0.16	12	78.13	+1.73	7	74.57	-2.90	5	79.20	+3.13	3
96	100	+4.17	9	89.29	-6.99	6	89.49	-6.78	4	105.60	+10.00	2
300	333.33	+11.11	2	312.50	+4.17	1	447.44	+49.15	0	316.80	+5.60	0
500	500	0	1	625	+25.00	0	447.44	-10.51	0	NA	-	-
HIGH	1000	-	0	625	-	0	447.44	-	0	316.80	-	0
LOW	3.91	-	255	2.44	-	255	1.75	-	255	1.24	-	255

BAUD RATE (Kbps)	Fosc = 4 MHz			3.579545 MHz			1 MHz			32.768 kHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	NA	-	-	NA	-	-	0.30	+0.16	207	0.29	-2.48	6
1.2	1.20	+0.16	207	1.20	+0.23	185	1.20	+0.16	51	1.02	-14.67	1
2.4	2.40	+0.16	103	2.41	+0.23	92	2.40	+0.16	25	2.05	-14.67	0
9.6	9.62	+0.16	25	9.73	+1.32	22	8.93	-6.99	6	NA	-	-
19.2	19.23	+0.16	12	18.64	-2.90	11	20.83	+8.51	2	NA	-	-
76.8	NA	-	-	74.57	-2.90	2	62.50	-18.62	0	NA	-	-
96	NA	-	-	111.86	+16.52	1	NA	-	-	NA	-	-
300	NA	-	-	223.72	-25.43	0	NA	-	-	NA	-	-
500	NA	-	-	NA	-	-	NA	-	-	NA	-	-
HIGH	250	-	0	55.93	-	0	62.50	-	0	2.05	-	0
LOW	0.98	-	255	0.22	-	255	0.24	-	255	0.008	-	255

The digital sequence between the clock cycles and the USART are shown in Figure 20.

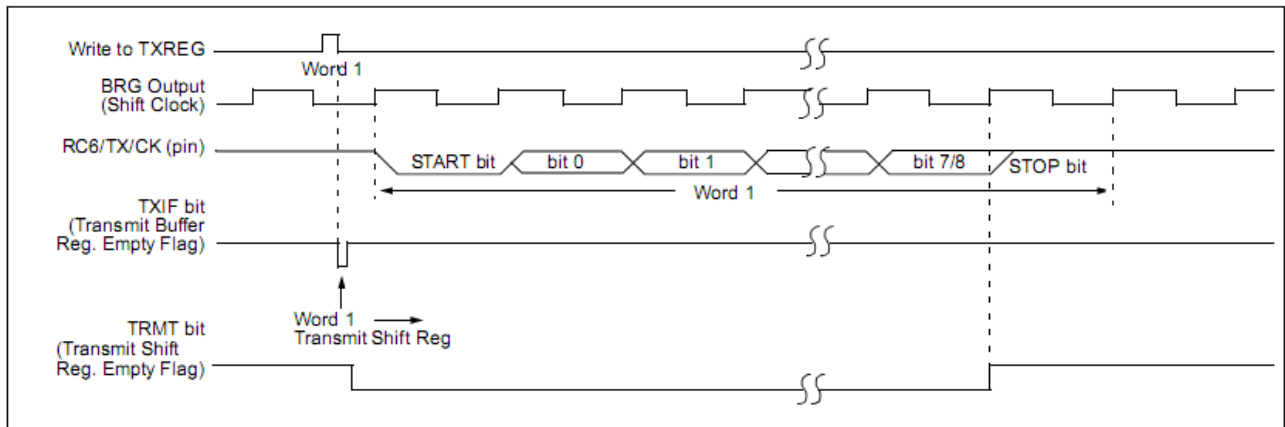


Figure 20: Digital Sequence between Clock Cycles and USART

As shown, there is a start bit and a stop bit associated with the USART. Also, with the USART it will output the least significant bit (LSB) first and the most significant bit (MSB) last.

Schematically, the PIC was configured as shown in Figure 21.

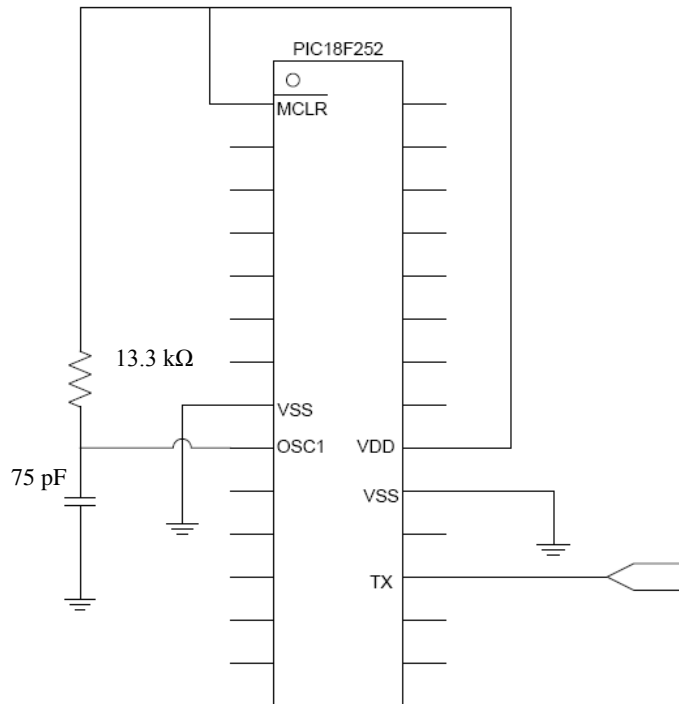


Figure 21: PIC Schematic

With the PIC programmed, tests between the connection of the TLP434 and RLP434 were conducted. The output of the RC6/TX pin and output of the receiver are shown in Figure 22.

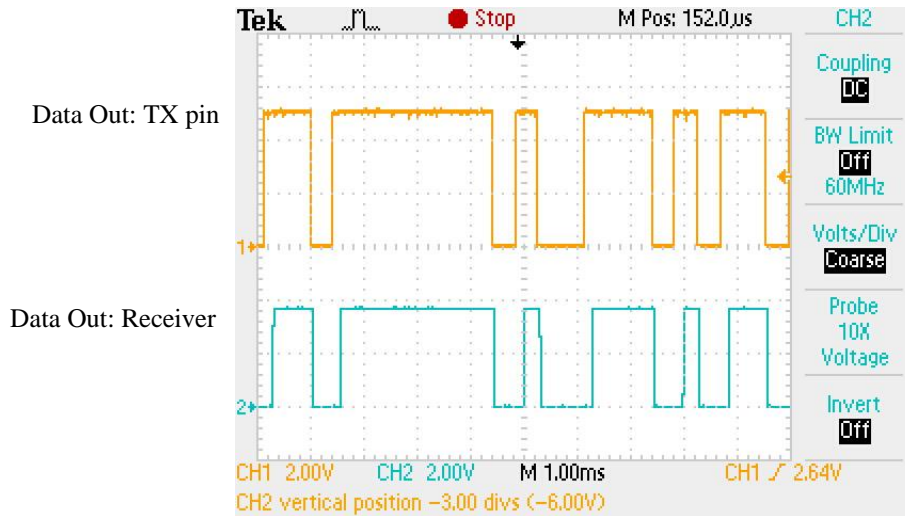


Figure 22: Output of RC6/TX pin and Receiver

The oscilloscope screen shot depicts the start and stop bits along with the ID number associated with the transponder with the LSB first and the MSB last. This evidence shows that the PIC is operating correctly. From this, functionality can now be added to the PIC in order to save on power consumption to meet the desired lifetime of the device. In order to do this, the Watchdog Timer and SLEEP mode of the PIC will be used. When using the SLEEP mode, the PIC shuts down most of its internal devices. Only the necessary components of the PIC remain operational; such as the clock. In order to wake the device up from SLEEP mode, you can either reset the Master Reset pin (pin1) or use the internal Watchdog Timer.

6.1.1.3 RF Link

Once the ID number data is outputted on the TX pin of the PIC microcontroller, it needs to be sent to the reader. This needs to happen when the patient is in range of the reader unit. Further details on this operation are described in the following section.

6.1.2 Receiver

The receiver takes the information sent out by the transponder and sends it to the nurses' station. This is done by using a receiver module and the Spartan3 board.

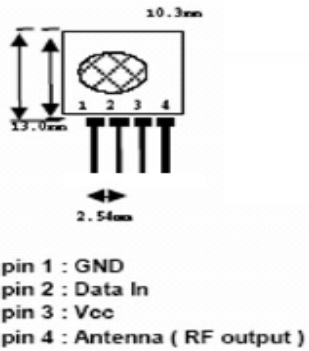
6.1.2.1 RF link

The operation that is fulfilled by the RF link of the system is the transmission of data from the patient to the doorway mounted receiver. The system requirements dictate that a serial string of data needs to be sent wirelessly from one unit to another over a minimum distance of 96 inches. Also, the power consumed by the receiver needs to be low in order to prolong the battery life while still accurately transmitting data.

There are a variety of RF modules available from linxtechnologies.com which were also considered for use in the project. Many RF modules were rejected for our purposes because the predicted ranges were approximately 100 meters, which is well beyond the minimum range of 96 inches. The system needs to avoid detecting the signal surpassing 96 inches to avoid the system constantly producing false alarms from transmitters that are not attached to people in danger of exiting the building. This particular RF link pair was used because members of the group had the module immediately available, and working knowledge of the components from previous endeavors.

This module consists of a radio receiver (RLP434A) which has been factory configured to receive Amplitude Shift Keyring (ASK) signals transmitted at 434MHz from the corresponding transmitter; the TLP434A. This component can be purchased as a package with the corresponding transmitter, and is designed to be a simple replacement for running a physical wire. The physical package has 8 pins; 3 of which are ground, 2 Vcc, 1 digital data output, 1 linear output/test pin, and an antenna pin. The configuration of the device involves attaching a wire to the antenna pin to serve as a rudimentary antenna. It also involves connecting a wire to the digital data output pin of the receiver, which is then connected to the Spartan3 board of the receiver, so the data can be transferred from the receiver to the Spartan3.

TLP 434, 434-A, 916-A



RLP 434, 434-A, 916-A, 916-F

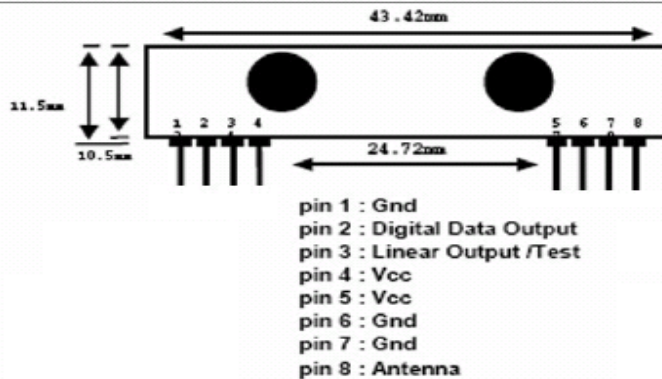


Figure 23: Tx/Rx Pair

6.1.3 Reader

After the information is transmitted from the transponder and read by the receiver, a reader is needed in order to interpret the data and send the information on its way to the computer interface.

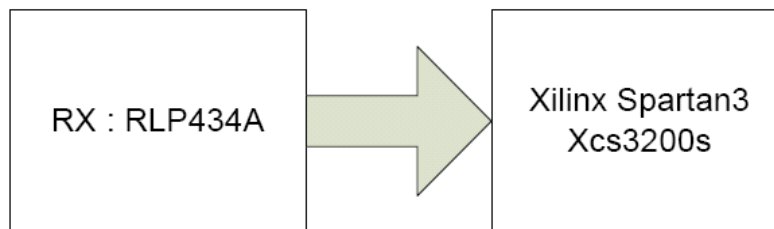


Figure 24: Receiver to Reader Block Diagram

6.1.3.1 Spartan3

The Spartan3 field programmable gate array (FPGA) needs to fulfill all the requirements that follow the wireless transmission of data in the above section. The Spartan3 must perform detection of an identification sequence and the data handling and manipulation that follows that detection. It must also configure the data to be sent serially out to the computer via a serial cable.

This functionality could have been implemented using any programmable logic device to detect the incoming signal and prepare it for serial transmission. However, it was decided to use as many familiar components as possible in the development of this prototype to reduce the

amount of new errors and coding systems that would need to be learned for the project to be successful. The Spartan3 development board was chosen for familiarity and the existing serial DB9 connection wired to the board.

The Spartan3 receives the serial transmission from the receiver through its transmission port. Using VHDL code, the reader constantly looks for a start code which signals that a patient's ID number has been successfully received. If the start code is correct the data bits of the transmission are sent to memory for storage. After storing the data, it is compared 3 times to eliminate noise from the system. After the comparison is complete the data is then sent to the serial port. If the start code does not match, the reader will continue polling the input. Figure 25 shows the reader flow diagram and Figure 26 displays the block diagram of the reader.

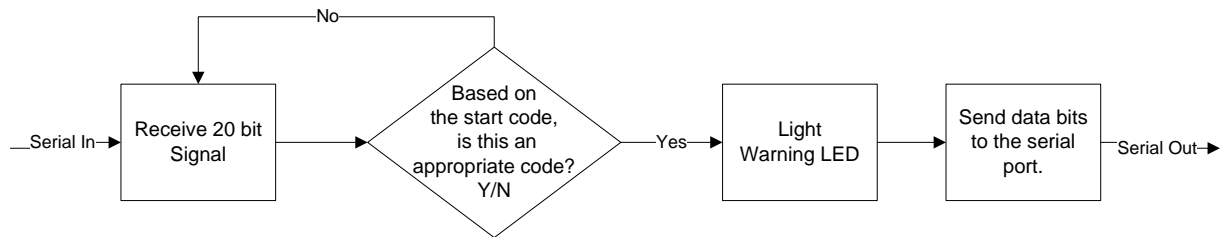


Figure 25: Reader Flow Diagram

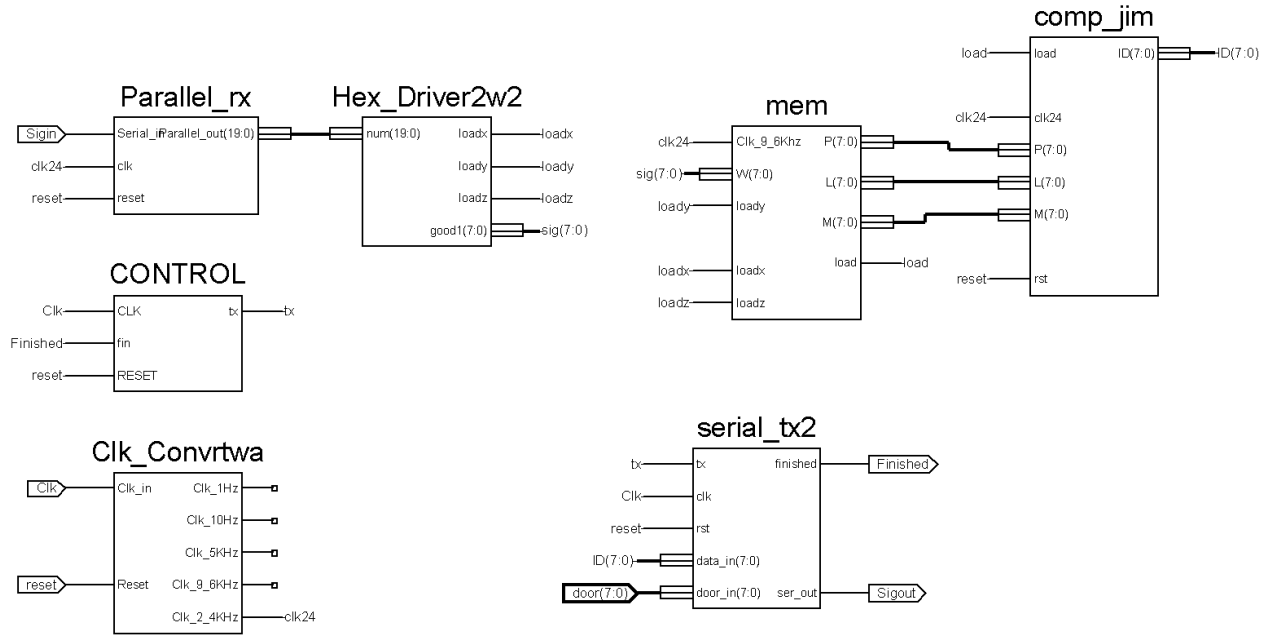


Figure 26: Reader Block Diagram

The code for the Spartan3 takes in a serial transmission from one of the I/O pins that exist on the board, and sends the data through the Parallel_rx block to transform the serial data into a 20 bit parallel configuration. The Parallel_rx module was done in VHDL. The code can be found in Appendix D: Spartan Code.

After the data is distributed into three variables, it is inputted into the comp_jim module. The comp_jim module was programmed in VHDL and the code may be found in Appendix F: Spartan Code. The first process this module does is to hold the values on the inputs P, L, and M. It first checks to see if the reset was pressed. If the reset was pressed then the module will reset. After it checks for the reset value, it checks to see if the load input is high or low. If the load input is low, then 0d is moved into the temporary variables. If the load input is high then the data on the input pins P, L, and M are stored into the temporary variables P_temp, L_temp, and M_temp respectively. This is so the data can be compared.

After this process is complete, it compares the temporary variables to see which value occurred most frequently. Again it checks for reset to be high. If reset is high then 0d will be found on the output. As long as load is high, it first compares P_temp and L_temp. If these are equal, then either P_temp or L_temp can be saved to PL. As the code is written, P_temp is saved to PL. PL is another temporary variable to store the second step of the compare process. If they are not equal then PL is set to 0d. After this comparison it goes

through the other comparisons in a similar fashion and saves them accordingly. After `PL`, `PM`, and `LM` are saved based on the initial comparisons, as long as `PL`, `PM` or `LM` are equal to one another, the value is deemed correct. The next stage in this process compares `PL` and `PM`. If they are equal then `PL` is saved to `good`. If they are not equal then `good` is set to `0d`. `Good` is a temporary variable that temporarily holds the correct data until it is saved to the output `ID`.

The next block is the `Serial_Tx` block. This block takes the output of the `comp_jim` block; `ID`, and prepares it for transmission through the RS232 port or serial port. This module also includes the door number which will be sent through the serial port as well. The first step in the module is to take in the `ID` number from the previous block and add a start bit, a parity bit, and two stop bits which prepare it for serial communication. It sets the parity by XORing the data bits, this determines whether there is an even or odd number of binary '1's. Since the parity is set to even, an even number of binary '1's is desired and the parity bit will change depending on the data. The same process is done for the door number. The door number is identified by the switches on the Spartan3 development board.

Once the 24-bits of data are configured as required by the RS232 port, the baud clock is configured. In order for the 2400 baud clock to be configured properly, the 50MHz clock the Spartan3 board runs on must be divided by 2400; yielding 20833. This value must also be divided by 2 which yields 10417. The value 20833 must be divided down again because the clock goes both high and low; thus will be high for half the time. A counter is implemented to index when the clock needs to reset.

After the baud clock is generated, the `continue` process is run. The `continue` process allows the data to be sent in its entirety. The first step is to check if the `continue` variable is high. If this is true then there is no need to continue with this process. If the `continue` variable is low, then this process checks to see if the `Tx` input is high. If the `Tx` input is high, then `continue` is set to high and the data will continue to be transmitted. If the `Tx` input is low, then the process checks to see if `finished_temp` is high. If this variable is high, then the data is done being transmitted and `continue` is set to low. If `finished_temp` is low, then `continue` retains its value and moves onto the next process; `transmit`.

The `transmit` process first checks to see if `reset` is high. If this is the case then the process is reset to its default values. If `reset` is low, on the rising edge of the clock and as long as `continue` is high, the values set previously on the variable `tx_data` are sent to the output.

This is done by using a counter. The `tx_data` is sent out bit by bit depending on count. When the counter hits 24, it sets `count` to 0, `finished_temp` high, and sets the output to high. If `continue` is low, then `count` and `finished_temp` retain their previous values. The output is then a 24-bit serial stream that is sent through to the computer interface.

The `Serial_Tx` module is controlled by the `CONTROL` module. This module sets the `tx` input depending on the state of the `Serial_Tx`. Figure 27 shows the state machine for the `CONTROL` module.

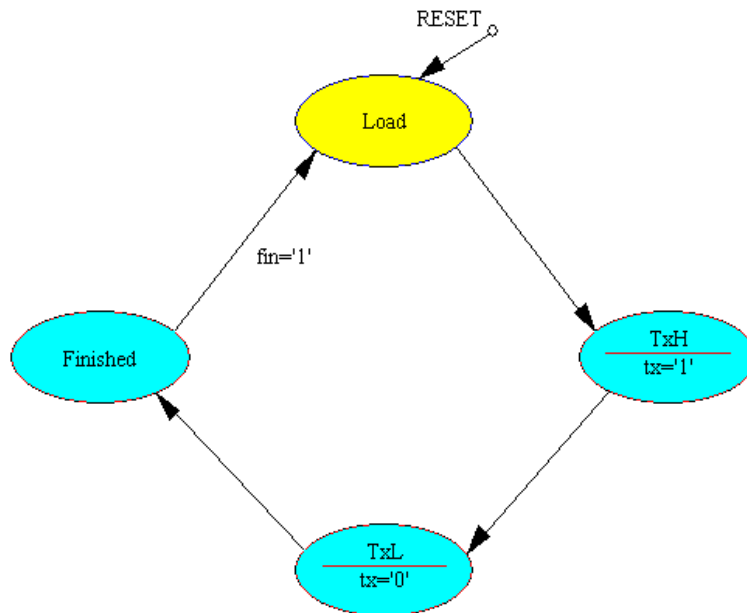


Figure 27: CONTROL Module

The purpose of this control is to set the `tx` pin high and low quickly. This is needed because the `tx` input into the `Serial_Tx` module loads the data into the rest of the processes and will set the `continue` variable. The first step in the state machine is when the reset input is high which will cause the system to start over at the load state. The load state is just a default state to which the system can return. On the next clock cycle, the state machine moves to the `TxH` state. This state sets `tx` high. `Tx` is set high for only one clock cycle since the very next state is `TxL`. On the next clock cycle, the state machine moves to the `Finished` state. This state waits until the input, called `fin`, goes high and then the process starts over again. This state machine runs on the internal clock of the Spartan3 board.

6.1.4 Computer Interface

The computer interface is located at the nurses' station, and is where the data received by the reader is sent to alert the staff of wandering patients. The use of an RS232 connection and a computer allows for this to be possible.

6.1.4.1 RS232

Once the data is processed by the reader, the ID number needs to be sent to the nurses' station to alert the nursing staff of the wandering patient. The operation that takes place at the nurses' station needs to be able to look at an Excel spreadsheet with all the ID numbers and the associated names. When the name associated with the ID number received is found, it needs to show a popup message alerting the nursing staff of the wandering individual. It also needs to initiate an alarm to attract attention. An assumption that all nursing stations contain a computer leads to the conclusion that the best option for this operation was using the computer.

A computer has the ability to retrieve information using a variety of interfaces. A computer can retrieve information from an Ethernet cable, a modem, USB device, Parallel port, Serial Port, CD ROM drives, and other such devices. The Spartan3 board also has numerous I/O ports that can either receive or transfer information to other devices. The RS232 or serial port was a common feature between the Spartan3 board and the computer. Sending information from the Spartan3 board via the serial port would be a feasible task. The more difficult process would be coding the computer to accept the incoming data and to process the information.

Foremost, the data needs to be transmitted from the Spartan3 board to the computer through the RS232 port. RS232 communication is a simple method of transmitting data from one point to another. The data being sent through the port needs to be configured in the same manner on both the transmitting and receiving ends to ensure correct transfer of data. First, the data being transferred needs to have a start bit; logic '0'. Next, the data bits are transferred. The RS232 port uses a parity bit as an error correction procedure; which is not a necessity, but is an option to add on after the data bits. A parity bit can be configured as even, odd, space (logic '0'), or mark (logic '1'). The stop bits are the final bits attached after the optional parity bits and are logic '1'. There can be 1, 1.5, or 2 stop bits.

Figure 28 [20] depicts how data is sent through the RS232 port.

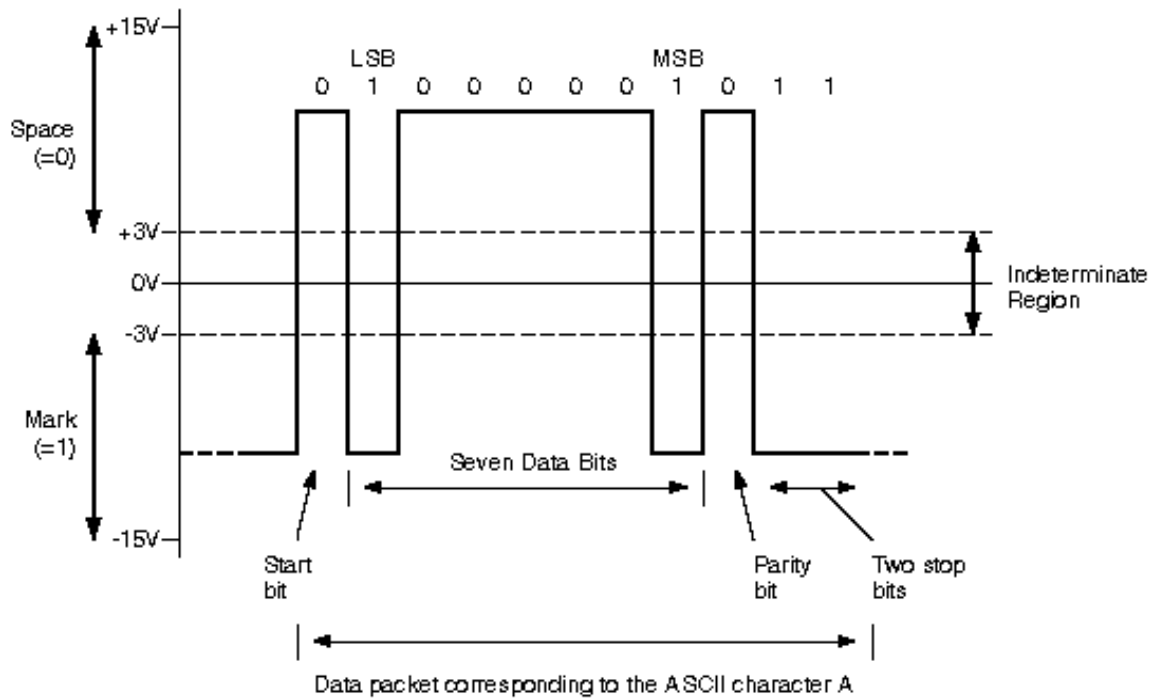


Figure 28: RS232 Data Byte Sequence

As mentioned previously, the start bit is logic '0'. In RS232 data logic '0' is +3V to +15V, whereas logic '1' is -3V to -15V. The Spartan3 board has a chip that converts the TTL level data to be converted into RS232 format. The computer has internal components that allow the incoming data to be converted from RS232 format to TTL level logic. If the parity bit is configured as even (even parity), the bit can either be a logic high or low depending on the data being transferred. If the data has an even number of logic '1', the parity bit will be logic '0'. If the data has an odd number of logic '1', the parity bit will be logic '1'. If the parity bit is configured as odd (odd parity), the bit will be a logic '1' where there are an even number of logic '1', and will be a logic '0' when there are an odd number of logic '1'. In the space configuration (space parity), the parity bit will always be logic '0'. If the bit is configured to be mark (mark parity), then the parity bit will always be logic '1'.

The computer is coded to accept an incoming serial bit stream. A copy of the code used is in Appendix I. The first step is to configure the RS232 port and open the file. Using the class made by serialport.h and serialport.cpp files (Appendix J), reading the serial port was possible. Since the data being sent from the Spartan3 board is 8-bits and has one start bit, even parity, and two stop bits, the serial port was configured in the same manner. Also, the RS232 port has to be

on the same baud rate as the Spartan3 board; which was set to 2400bps. After the RS232 port was opened and configured in the correct manner an array was created and initialized to store the data received by the RS232 port. After this array was created, the receiver buffer needed to be cleared in order to avoid inaccurate data reception. Next, the RS232 port needed to be read. The read function needs to know where to store the data received and how many words to read. For this project the buffer to store the data was the array created and wanted only 1 word (8-bits) to be stored.

Once the data was in the array, it could be processed. First the computer takes the values in the array and compares them for consistency. For testing purposes, a function called `SaveSP` was created to store each value in the array to calculate the bit error rate. This function however is not used by the customer. Another function called `ReadPID` was created in order to take the ID number received by the serial port and compare it to a list of ID numbers in an Excel spreadsheet. In order for the data to be saved to an excel file, the class `CSPreadSheet` was implemented (Appendix K). Once the ID number received by the serial port was matched to its corresponding value in the Excel spreadsheet, it returned the name associated with the ID number. After this is returned, the computer speaker sounds an alarm while a popup message is sent to the computer screen warning the nursing staff of the wandering patient. The popup window contains the name of the individual that wandered off along with the associated door number that the patient exited. This popup message was created using the Visual Studio C++ Message configuration wizard. The code for the popup message can be found in Appendix L. Figure 29 shows the pop-up message and computer program running.



Figure 29: Computer Pop-Up Alarm

This program is constantly running as long as the computer is powered and running. This ensures that monitoring is continuous. Once a patient has been detected, a nurse has to hit the 'OK' button in order for the code to continue to run and the reset button on the reader also needs to be pressed. If the reset button on the reader is not pressed, then the computer will continue to display the message until the user resets the reader.

7.0 Design Testing

The various components described above in detail are connected in the following manner to complete the system:

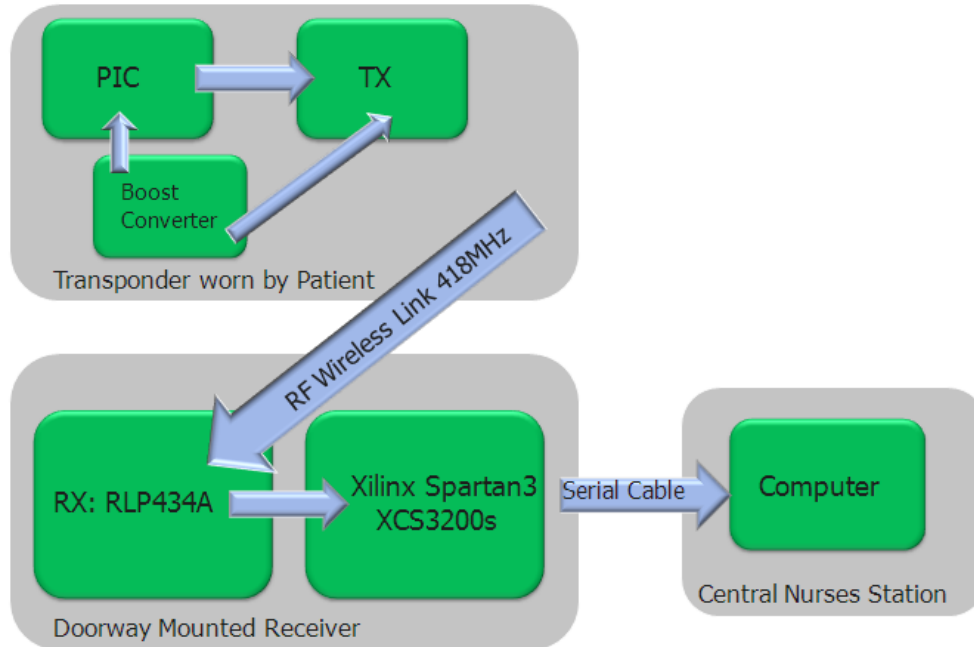


Figure 30: Design III - Detailed Block Diagram

7.1 Transponder

The first component tested was the boost converter. With the configuration mentioned above, the output voltage ripple is 32mV. This is within the specifications the datasheet gives us of 40mV. Figure 31 shows the output voltage ripple of the boost converter.

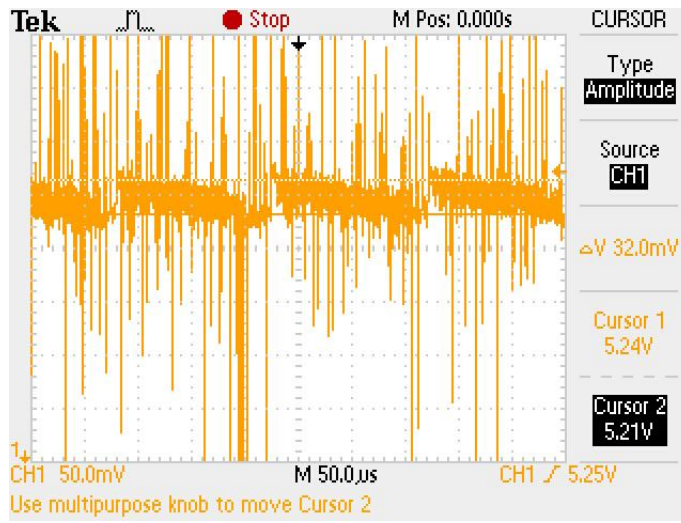


Figure 31: Boost Converter Output Voltage Ripple

Within the transponder a PIC connects to a transmitter (TLP434A RF link) that transmits Amplitude Shift Keying (ASK) modulated data at 434 MHz to its receiver pair, the RLP434A. This transmitter/receiver pair only requires a digital signal input, V_{cc} , ground, and the output to the antenna on the transmitter side. The receiver has 3 ground pins, a digital data output, $2V_{cc}$ pins, a linear data output, and the antenna input.

The ID number the PIC was programmed to store successfully transmits the information continuously to its transmitter. Figure 32 shows the output of the PIC microcontroller.

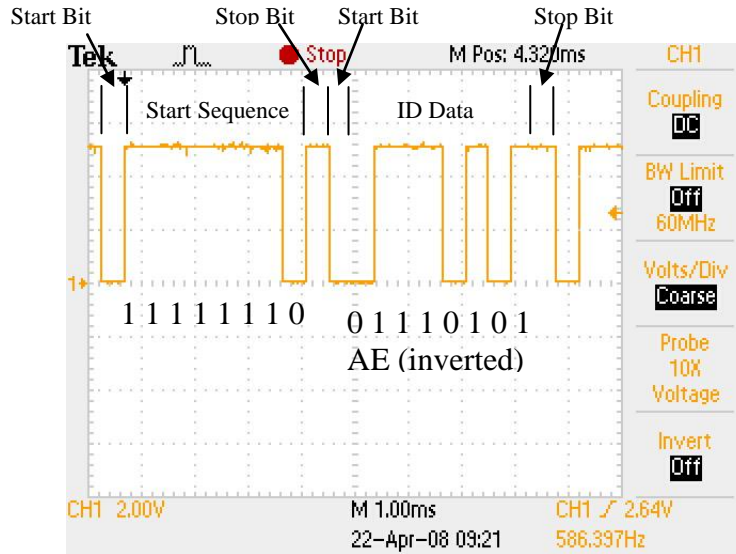


Figure 32: Data Out from PIC Microcontroller

As can be seen, the first part of the data contains a start bit, then the start sequence data, a stop bit, start bit, ID data, stop bit. We then took a look at the bit length to determine the frequency at which the signal is being sent. We expected that the bit length should be around $400\mu\text{s}$ because of the baud rate of 2400bps . Figure 33 shows the bit length from the output of the PIC



Figure 33: Bit Length of One Bit of PIC Output

The measured bit length is $428\mu\text{s}$ which is 6.52% higher than the estimated $400\mu\text{s}$. Once the data out of the PIC was confirmed, the receiver was then tested.

The receiver has been tested and was able to receive the information from the PIC and is able to communicate the data to its receiver pair; bridging the gap wirelessly between the transponder and the receiver. Figure 34 shows the output of the PIC and the receiver output.

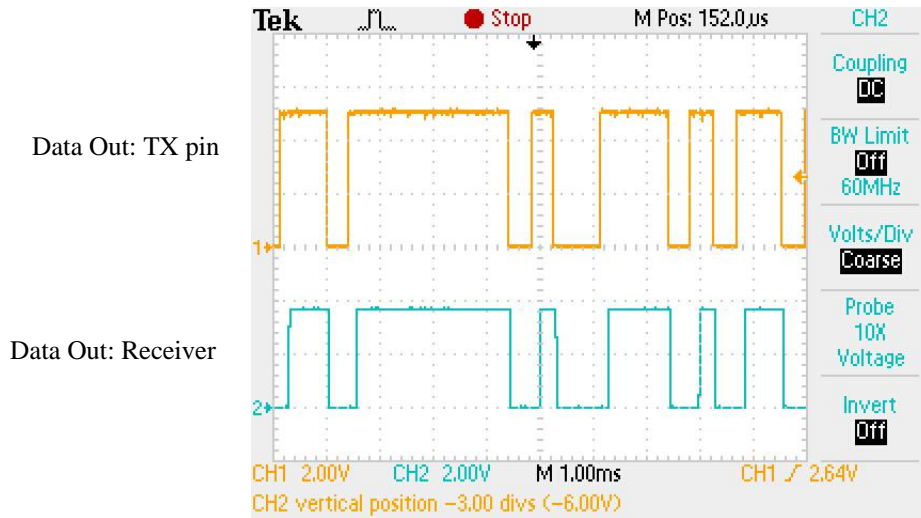


Figure 34: Transmitter Input/ Receiver Output

We then looked at the time delay between the output of the PIC microcontroller and the output of the receiver. Figure 35 shows the time delay between the transmitter and receiver over 1 meter.

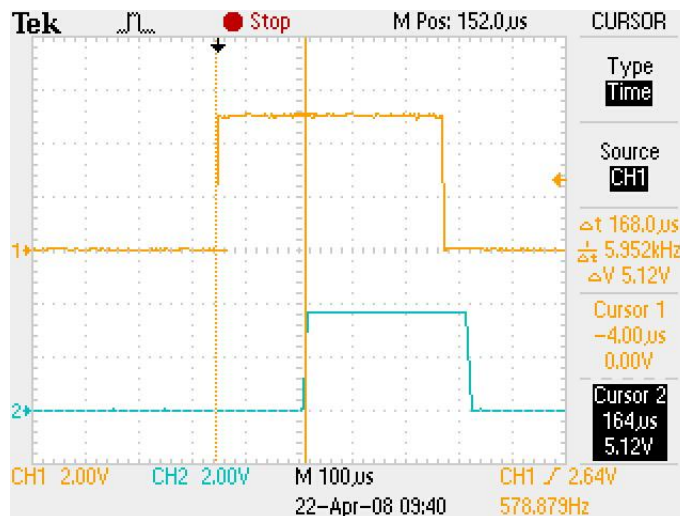


Figure 35: Transmitter and Receiver Time Delays

We then took a look at the bit length of the receiver. We estimated that this should be about the same as the output of the PIC. Figure 36 shows the bit length of the receiver output.

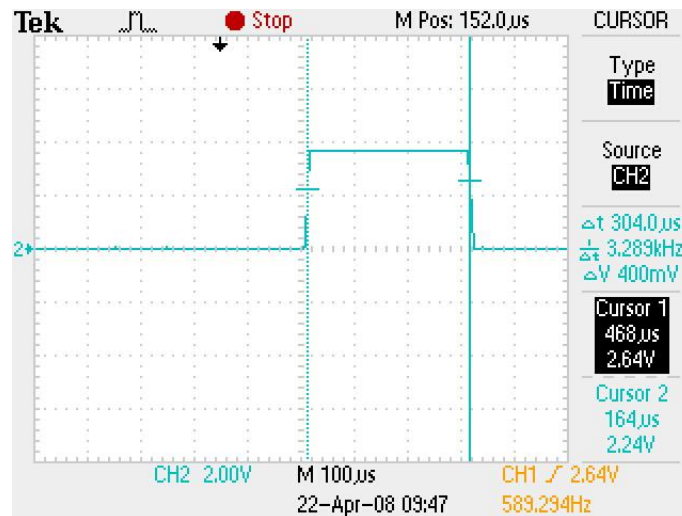


Figure 36: Receiver Output Bit Length

We measured that the bit length was 304 μs which is roughly 100 μs less than the output bit length of the PIC. This could cause problems when trying to decipher the signal coming in and may add or remove bits depending on the timing.

7.2 Receiver

The digital data output from the RLP434 connects through an I/O port on the Spartan3 FPGA. The Spartan3 connects through a serial cable to the RS232 serial port of a computer.

The receiver pair of the transmitter successfully receives the information transmitted wirelessly from the transmitter. However, once that information is collected and then transferred to the Spartan3 via a serial cable, errors within the code made it unable to authenticate the signal 100% of the time.

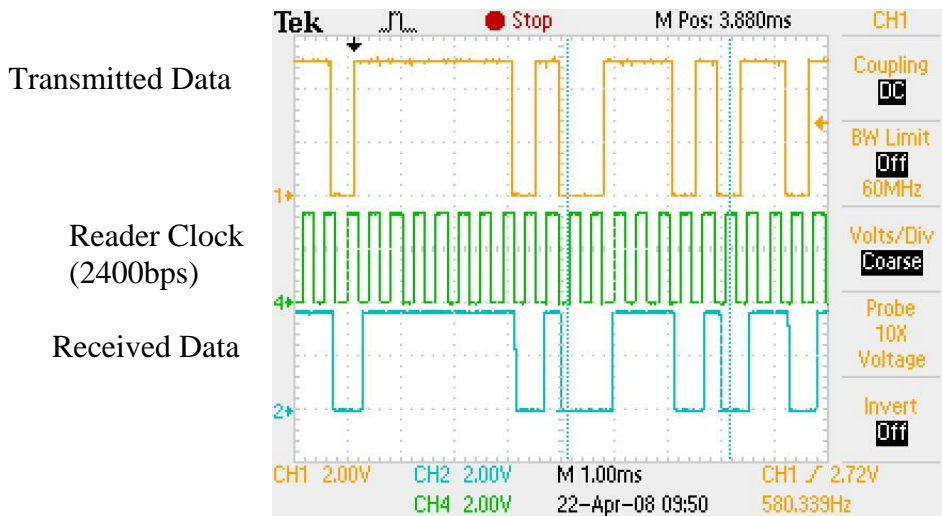


Figure 37: Tx, Rx and Reader Clock Waveforms

When looking at the reader clock compared to the transmitted data and received data, it is easy to see where problems could have and did arise. The problem is trying to synchronize the data coming out of the PIC to the received signal. The reader clock is matched up to the transmitted data. However, since the receiver bit length is 100 μ s less than the transmitted bit length, the timing on the received data is slightly off causing errors.

7.3 Central Nurses Station

The computer located at the central nurses station has a database that contains the patients' name, and matches it to the ID number transmitted by the transponder. The computer then displays the patients name and the doorway that the patient has tried to go through in a flashing box on the screen alerting the staff of the occurrence.

When testing the computer interface to see if it could locate and alert the staff via a pop-up message on the computer, the program worked successfully.

7.4 Analysis

One of the more trying tasks of this project was to try to synchronize the transponder clock to the reader clock. This was difficult since the transponder's clock is asynchronous where as the reader's clock is synchronous. However, in designing a UART to handle the incoming signal and then compare it to a known sequence, we could get the correct ID number. After the

system was officially working, some tests on the probability of reading the correct signal based on distances were performed. The serial port was read 1000 times over a distance of 1 meter, 2 meters, 5 meters, and 10 meters. For a reading distance of 1 meter we found that the correct ID number was read 741 times. This means that the probability of detection is 74.4%. For 2 meters we found that this number was 697 times meaning a probability of detection is 69.7%. For 5 meters, the probability of detection decreased to 63.1% or 631 times detected out of 1000. And finally for 10 meters we found that correct detection occurred 751 times or 75.1% probability of detection. Table 8 shows the tests done on distance.

Table 8: Distance Tests

	1 meter	2 meters	5 meters	10 meters
Times Detected	741	697	631	751
Probability	74.1%	69.7%	63.1%	75.1%

This receiver is capable of transmitting a signal up to 350 feet. Ideally we would want the range of detection to be at a max of 2 meters. This would prevent detection of patients that are still in the nursing home facility and have no desire to wander away. A desired probability of detection would be 90% or greater; but due to the inability to properly synchronize the clocks, the probability is considerably lower.

8.0 Conclusion

After pursuing a couple designs, it was found that more requirements needed to be defined in order to better choose components that would operate to the desired characteristics. Upon this change, a new design followed with stronger guidelines as to what the system needed to do in order to operate and what stipulations certain components needed to follow in order to perform these operations.

With a proper system design document supporting the design choices, a new system was developed and tested. Each component was tested individually to ensure it was operational before attempting to connect it to its subsequent component.

Throughout the course of this projects endeavor, various purchases of components and devices to aid in our design and testing purposes were crucial. Although a hypothetical value analysis on production costs and projected return on investments were this device to be marketed was previously analyzed; a realistic analysis on what finances were incurred is described in the following section.

8.1 Finances

In order to assist students financially in completing major qualifying projects (MQP) within the Electrical and Computer Engineering (ECE) department, each student is allotted \$75.00; regardless of team size. Given that our project team consists of four individuals, we have been allocated a budget of \$300.00 from the department. This \$300.00 is to assist us in ordering components or any other devices that cannot be found within the department's laboratory facilities to aid in our design and testing.

Ordering components for our project can either be done through the department's shop where the money comes directly out of our budget, or by purchasing things needed at our own expenses for which we are reimbursed at the completion of the project provided proper documentation is presented (i.e. receipts).

When ordering components through the department's shop, an order form complete with group contact information, vendor, vendor part number, cost, quantity needed, and final costs are submitted.

MQP Digikey Order Date: 11/27/07 Rev A			Group: Students:	SaunterNot Edmund Massa Stacey Mohr James Medeiros William Brooks Prof. Labonte	perloc08@wpi.edu wopsicle@wpi.edu samohr@wpi.edu jimbo08@wpi.edu wbrooks4@wpi.edu rcl@ece.wpi.edu	
			Engineering Manager:			
Item	QTY	Part No.	Description	Value	Unit	Sub
1						
2						
3						\$0.00
				Digikey Total		\$0.00

Figure 38: Parts Order Form

Records of each order form that have been submitted have been documented and made easily accessible by all personnel in the project group. This has made it considerably simple to keep track of our finances to make sure we do not go over our budget.

Compiling our ECE lab kits that were distributed to every student who took the Introduction to Electrical Engineering course at WPI has been a useful way to help keep our costs down. In each of our kits we are supplied with a breadboard, variety of different resistor values, various capacitors, transistors, LM555 timers, and LEDs; all of which have been used in our prototyping.

When parts were not available through our lab kits, we tried to either order complimentary samples, purchase them at an in store location, or order them through a vendor.

Ordering complimentary samples was easily done primarily through Texas Instruments (TI). When looking up a component on the TI website, if it is available as a complimentary sample, they have a link next to the pricing information section. From this link you can fill out an order form online to have the product shipped to you free of charge. What was found to be a quicker more efficient way of ordering complimentary samples was to use their “Get Samples” link found on their homepage. From here TI has a diverse list of component applications one may be searching for and from there applicable components. Once the desired component was found, calling your local sales representative and speaking to someone directly proved to be an efficient way to not only order a complimentary sample but to opt for the sample to be shipped overnight if needed.

Complimentary samples received from TI were our 22mm transponder we first used to test with our reader, and our shift register; which they graciously shipped overnight. In both instances TI supplied us with 10 samples, for which we can donate our unused components to the

ECE lab in case other students are looking for a similar product. Through TI we were able to save approximately \$30.00 between the two components.

RadioShack was found to be a viable in store location for our electronic needs. For instance, we purchased an AC/DC converter and component that connected the power converter to the breadboard from RadioShack. RadioShack also provided our casing for the reader and the battery holder used to connect to the breadboard for our transponder prototype in Design III. Buying the devices at an in store location was preferable to ordering them through a vendor since we did not have to wait for the items to be shipped nor did we have to pay extra shipping and handling fees.

However, for more specific applications that we have tried to achieve in our testing required different components which are not sold at in store locations. Such components include a DC/DC converter, analog switch, RFID transponder inlay tag, a decoder/demodulator, and boost converter. These components were ordered through the ECE department lab by filling out the order forms as depicted below.

MQP Digikey Order Date: 11/27/07 Rev A			Group: Students:	SaunterNot Edmund Massa Stacey Mohr James Medeiros William Brooks Prof. Labonte	perloc08@wpi.edu wopsicle@wpi.edu samohr@wpi.edu jimbo08@wpi.edu wbrooks4@wpi.edu rcl@ece.wpi.edu	
			Engineering Manager:			
Item	QTY	Part No.	Description	Value	Unit	Sub
1	2	FSA2257L10XCT-ND	IC SWITCH ANLG DUAL SPDT 10MCRPK		\$0.99	\$1.98
2	3	481-1078-1-ND	RFID TRANSP IN-LAY GEN II 1X3.5"		\$1.19	\$3.57
3						\$0.00
				Digikey Total		\$5.55

MQP Mouser Order Date: 11/27/07 Rev A			Group: Students:	SaunterNot Edmund Massa Stacey Mohr James Medeiros William Brooks Prof. Labonte	perloc08@wpi.edu wopsicle@wpi.edu samohr@wpi.edu jimbo08@wpi.edu wbrooks4@wpi.edu rcl@ece.wpi.edu	
			Engineering Manager:			
Item	QTY	Part No.	Description	Value	Unit	Sub
1	2	513-NJM#2211M	Decoders Demodulator/Decoder		\$1.24	\$2.48
2						
				Mouser Total		\$2.48

Figure 39: November 27, 2007 Order Form

MQP Digikey Order		Group:	SaunterNot	perloc08@wpi.edu		
Date: 1/11/08		Students:	Edmund Massa	wopsicle@wpi.edu		
Rev A			Stacey Mohr	samohr@wpi.edu		
			James Medeiros	jimbo08@wpi.edu		
			William Brooks	wbrooks4@wpi.edu		
		Engineering Manager:	Prof. Labonte	rcl@ece.wpi.edu		
Item	QTY	Part No.	Description	Value	Unit	Sub
1	1	102-1364-ND	DC/DC Converter		\$4.66	\$4.66
2						
3						
				Digikey Total		\$4.66

Figure 40: January 11, 2008 Order Form

MQP Digikey Order		Group:	SaunterNot	perloc08@wpi.edu		
Date: 04/02/08		Students:	Edmund Massa	wopsicle@wpi.edu		
Rev A			Stacey Mohr	samohr@wpi.edu		
			James Medeiros	jimbo08@wpi.edu		
			William Brooks	wbrooks4@wpi.edu		
		Engineering Manager:	Prof. Labonte	rcl@ece.wpi.edu		
Item	QTY	Part No.	Description	Value	Unit	Sub
1	4	296-3229-5-ND	IC 5V SYNCH BOOST CONV 8-DIP		\$4.05	\$16.20
2						
				Digikey Total		\$16.20

Figure 41: April 2, 2008 Order Form

The three primary vendor companies the ECE lab orders from are Digikey, Mouser, and Newark. When ordering parts from one of these vendor companies, they only ship parts if the total cost is at least \$15.00. Otherwise fees are added to the total order. This was not a concern during November, December, March and April when the ECE design course was in session and many students were placing orders to one of these three companies. With orders going out on a weekly basis we were able to easily be added to the shipment increasing the total cost of the order to be placed. However, come January fewer orders were being placed to these companies from the ECE lab. Because of this, we had to incur additional charges to meet their minimum price per order.

However, one thing we found to be beneficial when ordering parts was to order at least two of any part we needed if the cost was relatively inexpensive (under \$5.00). In doing this, we made sure that were one part to break or not function properly during testing, we would have a back-up part to work with for which we did not have to wait for the turnaround time or pay the extra fees for ordering extra parts.

9.0 Recommendations

The final design pursued was completed as a ‘proof of concept’. Given the various detours we had taken in attempting other designs, time became an issue. With the deadline drawing nearer, creating a product that would operate as a ‘proof of concept’ and could later be altered to be marketable was our best option. Being a ‘proof of concept’ and not designed specifically for one marketplace or use, several alternative uses for the concept have also been addressed in the following section. Along with alternative design uses, additional options for future projects to add to the design are also addressed.

9.1 Device Marketability

Given various discrepancies that do not qualify this design to be marketable, it is designed as a theoretical product for the testing of what could later be made a marketable product. In order for the product to be marketable, it would have to endure heavy testing to prove it’s reliable in various environments, the system would need to have a stronger security for the wireless data transmitted, and the design of the transponder would need to be altered for specific applications.

Without the product operating around a reasonable 99% of the time (since nothing is ever perfect) the safety of the patient wearing the device would be in jeopardy. The intention of this product is to put the staff and family at ease that there is a constant monitoring system in place ready to alarm if the patient tries to leave the building. Were this system to fail, it then puts the patient at risk of wandering off into a hazardous situation without anyone knowing. If the staff becomes dependent on the system and neglects to periodically check on the whereabouts of the patients, they could go missing for a longer duration of time allowing for more distance to get between them and the facility. Because of these possibilities, intense testing on the reliability of the system would need to be performed before attempting to make the device marketable.

Currently, the information being transmitted wirelessly from the detection of the transponder to the receiver contains a number used to associate with the name of the individual wearing the transponder. The transponders 8-bit identification number is transmitted to the receiver when the transponder comes within range of the receiver. With the name that is sent to the nurses’ station, the nurses can then look up the individuals information if need be in order to

find out any health issues that may be pertinent to locating the individual. However, in a more intimate setting, the nurses would be able to associate the individuals name to their various tendencies hopefully cutting down on search time. For instance, if John Smith is known to like to smell the flowers outside, when the alarm triggers that ‘John Smith has exited door 12’, the staff can first look towards the flowers outside to see if he is in the area.

Given that information is transmitted wirelessly, in order for the product to be marketable, we would need to ensure security measures are taken so that an outside party is not able to intercept the wireless signal and obtain the information being transmitted.

In addition to stronger wireless security, disclosure forms would also be signed by the individual’s family or guardian acknowledging that they are aware of what information will be transmitted and that they are aware of the precautions being taken to secure such information. Disclosure forms would also need to be used so that the families or guardians of the patients are aware that the purpose of the device is not a tracking unit, however is merely an alarm if they leave the building. This would help to eliminate the fear that the patient’s privacy is breeched and enlighten that the device is only meant to help the staff ensure the patient’s safety.

Another alteration that would need to be done in order to make the device marketable for the purposes of patient monitoring is to remodel the transponder. Currently, the transponder design is too large and bulky for a patient to comfortably wear. Newer and more advanced technologies than the ones we pursued that would make the device smaller and lighter while still maintaining dependability would be ideal.

9.2 Alternative Design Uses

This RFID system has a variety of uses for which it can easily be adapted. In general, RFID systems are used for many applications in areas such as automotive, animal tracking, asset tracking, contactless commerce, and in retail stores. Our device can be used for many of these applications with minor software related adjustments and with modifications to the size and/or weight of the transponder. Two ideal applications for our device would be asset tracking and retail use.

Libraries and hospitals depend on their books and equipment so that everything runs smoothly. Asset tracking using RFID would assist hospitals not just with patient tracking, as has

been discussed, but also tracking equipment and supplies. This application would be useful to help locate equipment that has been moved through certain doorways and placed in other rooms. This could also reduce the chance of any equipment being stolen. Libraries could also benefit from an RFID system to track books or supplies and reduce the amount of missing or stolen resources.

Retail stores could utilize the system for several applications that could reduce stolen goods and keep track of their stock. Many retail stores have theft reducing measures, but our RFID system has the unique ability to identify what is being moved and what doorway it is being moved through. This could efficiently keep track of what stock goes from the stock room to the shelves; or what never makes it to the shelves. The system could also electronically keep track of inventory as it arrives.

Museums would benefit greatly from this system as an information feature in case some of their security measures fail. The transponders could be programmed to identify which piece of artwork or relic they represent so that in the chance the piece is stolen, the museum would be able to identify which piece of work is exiting which door in real-time. This would add to the benefit of their alarm system currently in place which only sounds when the piece is being taken and provides minimal information on the location.

Other applications the system could be used for are amusement parks, stadiums, or venues where people have season passes. The season pass can take the form of a transponder tag and the individual can easily enter the park or stadium in the same fashion a car passes through an E-Zpass lane thus cutting down on lines. With this same concept, the system could be used for ski resorts as well. Rather than have people purchase day tickets or evening tickets, you could pay for the number of times you wish you use the chair lift. With this, every time the individual goes to use the chair lift, they would have to swipe their card which would subtract that ride from their total number of rides left.

Many project variations could arise from the basic concepts of the SaunterNot system that would be beneficial to its users

9.3 Future Projects

With there being many alternatives you could apply the concepts of the SaunterNot system to for alternate applications, there are also a variety of features you could add to the SaunterNot system given its originally designed application; for the detection of wandering patients.

One thing discussed when interviewing with Mrs. Cie Galloway during our preliminary research was that it would be beneficial to the nursing staff if the transponders were also equipped with a vital signs monitoring system. Although this would not take place of the nursing staff regularly taking patients vitals, it would be a beneficial feature to alert the staff if a patient is at a health risk. If a patient was alone in their room and either stopped breathing, had their blood pressure drop too low, or fell; some time may pass before a staff member or another resident finds them decreasing their chance of proper care and survival. With a vital signs monitoring system, the nursing staff would be alerted immediately if any of the previous were to happen, and could seek out that patient to give them the proper aid in a more timely manner.

Another feature that would be desirable for the transponder would be to have it go into a sleep mode when the patient is inactive. If the individual wearing the transponder is either asleep or sitting down, they are not at risk of exiting the building at that current moment in time. In these types of situations, it would be ideal for the transponder to turn itself off in order to conserve the battery. However, the difficulty would be in determining how the transponder would detect such instances reliably.

Appendix A: Interview with Cie Galloway

Meeting with Regional Clinical Assistant

Interim Director of Nurses – Cie Galloway

September 11, 2007

1:00 P.M. – 2:00 P.M.

Parson's Hill Rehabilitation Center

1350 Main Street

Worcester, MA

Interviewers Present

Brooks, Will

Massa, Edmund

Medeiros, James

Mohr, Stacey

Interviewees Present

Galloway, Cie

Meeting Agenda

The purpose of the meeting was to obtain information from a first hand source on their experiences of working with Alzheimer's and dementia patients.

Meeting Synopsis

Mrs. Cie Galloway is a Regional Clinical Assistant who is currently sitting in as an Interim Director of Nurses at Parson's Hill Rehabilitation Center. Prior to her experience with the Rehabilitation Center, Mrs. Galloway has worked at nine other nursing facilities – one of which includes experience with a geriatric-psyche ward in Lowell, MA. After interviewing Mrs. Galloway for an hour, the group developed a greater insight as to the various precautions the facility takes in keeping their patients safe, the mannerisms of various types of patients, and suggestions as to what would be most beneficial to their needs.

As a means to keep the patients from wandering off into a foreign area, the doors that close off the different wards and exit doors of the home are guarded by a keypad system. In order to get through the doors to a different section of the building, one would need to know the code. This however does become a problem when patients learn the code by watching a nurse punch it in, and then later use it to exit.

For a precaution on the windows in the nursing facility, the windows are kept locked at all times and only have the ability to open 6". The nursing facility is kept cool by a central air conditioning system during the summer months to avoid the heat; however window access is available per request.

At times patients do wander into different rooms, and are unable to be located by the nursing staff. However, when inquiring about the use of personal location devices that

monitor to the whereabouts of an individual at all times, the legalities of privacy impressed by the Department of Public Health were a concern.

There is also a device in place called the WanderGuard System that is installed at the building exit doors around the Alzheimer's and dementia ward. For this system, a candidate is chosen not based on their condition, but must fulfill a criteria assessment to determine whether or not they are at risk of wandering. In addition to this, the consent of the patient if they are still of able mind or of the family to allow the patient to take part in the system is needed. If the patient were to fulfill the requirements and be recommended as a candidate for the device, but the family objects to its use, then the facility is not to use the device on that patient. Some families feel that the device only causes more agitation for the individual wearing it when alarms are sounded and by its general aesthetics as shown in Figure 42 [4]. Many individuals try to remove the transmitter device that is strapped to their ankle due to its unfamiliarity and discomfort. The WanderGuard System works only through doors with the receivers installed. Each door receiver setup costs approximately \$3,000.00 and each transmitter device costs approximately \$75.00. The door setup consists of two 8" receivers, one on each side of the door, that alarm when the transmitter breaks the sensor beam between the receivers. However, given that the receivers are only 8" tall, patients can learn quickly how the system works and are able to exit the building by stepping over the beam to avoid the system. Another difficulty with the current device is that it does not inform the nurses who has exited the door. There is a common alarm for all individuals with the transmitters.



Figure 42: WanderGuard Transmitter Device

Given Mrs. Galloway's extensive experience in various nursing facilities, she was able to also speak on part of various patients' normality's. It depended strongly on the variety of patients within a facility whether or not there were more demanding issues of people wandering out the doors. In a geriatric-psyche ward, it was a daily occurrence. However, in a more relaxed environment such as that of Parson's Hill Rehabilitation Center, they have only experienced one individual wandering off since July 2007.

After inquiring more specifically about the needs of the facility and what she would deem most beneficial to the nurses on staff, several suggestions arose. To know not only that an individual has exited the building, but to know who has left would be one of the most beneficial attributes to a new system. In addition to this, a system that monitors individual's vital signs would be another valuable asset.

Appendix B: Resident Rights Category

Resident Rights Category

Criteria in the resident rights category concern quality of life and resident rights in a nursing home. A facility must care for its residents in a manner and in an environment that promotes maintenance or enhancement of each resident's quality of life.

1. The facility ensures protection of resident funds.
(42CFR 483.10(c) (1) F 158)
2. The facility honors the privacy and confidentiality of each resident.
(42CFR 483.10(e) F 164)
3. The facility ensures that residents are free from physical restraints used for purposes other than to treat the resident's medical symptoms.
(42CFR 483.13(a) F 221)
4. The facility develops and implements written policies and procedures that prohibit mistreatment, neglect, and abuse of residents and misappropriation of resident property.
(42CFR 483.13(c) F 224)
5. The facility promotes the quality of life for each resident.
(42CFR 483.15 F 240)
6. The facility ensures the dignity of each resident.
(42CFR 483.15(a) F 241)
7. The facility encourages residents to participate in the planning and decisions related to care and treatment, to make choices about aspects of his/her life in the facility that are significant to the resident.
(42CFR 483.15(b) F 242)
8. The facility provides each resident with the right to reside and receive services with reasonable accommodation of individual needs and preferences, without endangering the well being of the individual or other residents.
(42CFR 483.15(e) (1) F 246)
9. The facility provides an ongoing program of activities designed to meet the interests and physical, mental and psychosocial well being for each resident.
(42CFR 483.15(f) (1) F 248)

Source: <http://www.mass.gov/dph/qtool2/qtrights.htm>

Appendix C: Transponder Code

PIC Code

; Initialization Code for PIC18F252, Family: GP control, Package: 28-Pin SDIP 28pins

```
#include P18F252.inc
```

```
;equates
```

```
Bank0Ram    equ    0x20                ; start of Bank 0 RAM area
TenMsH      equ    2                    ; Initial value of TenMs Subroutine's counter
TenMsL      equ    70
```

```
;variables
```

```
    cblock Bank0Ram
        COUNTH                ; Two-byte counter for TenMs subroutine
        COUNTL
    endc
```

MainLoop

; Feature=IOPortA - IO Ports configuration

; port A is 6 bits wide

; set TRIS to all inputs before setting initial value

```
    movlw 0x3F
    movwf TRISA
    clrf PORTA
```

; set port bit as input (1) or output (0)

```
    movlw 0xFF
    movwf TRISA
```

; Feature=IOPortB - IO Ports configuration

; port B is 8 bits wide

; set TRIS to all inputs before setting initial value

```
    movlw 0xFF
    movwf TRISB
    clrf PORTB
    ;movlw 0xAE
    ;movwf PORTB
```

```

        clrf PORTB
; set port bit as input (1) or output (0)
        movlw 0xFF
        movwf TRISB

; Feature=IOPortC - IO Ports configuration

; port C is 8 bits wide

; set TRIS to all inputs before setting initial value
        movlw 0xFF
        movwf TRISC
        ;movlw 0x7F
        ;movwf PORTC
        clrf PORTC
; set port bit as input (1) or output (0)
        movlw 0xFF
        movwf TRISC

; RX/TX buffer
        movf SSPBUF, W
; Feature=UART1 - USART configuration

; (RCSTA)B7=SPEN B6=RX9 B5=SREN B4=CREN B3=ADDEN B2=FERR B1=OERR
B0=RX9D

; (TXSTA)B7=CSRC B6=TX9 B5=TXEN B4=SYNC B2=BRGH B1=TRMT B0=TX9D

; (SPBRG)Baud rate generator

; (RCREG)Receive register
        movlw 0x80                ; set up receive options
        movwf RCSTA
        movlw 0x26                ; set up transmit options
        movwf TXSTA
        movlw 0x19                ; set up baud
        movwf SPBRG
        movf RCREG, W            ; flush receive buffer

Trans
        BTFSS      TXSTA, TRMT    ; Checks to see if the TRMT bit is empty or '1'
        GOTO      Trans          ; goes to Trans if TRMT is not '1'
        movlw     0x7F           ; start sequence to synchronize
        movwf     TXREG          ; 0xCD moved to TXREG to be transmitted
Trans_1

```

```

        BTFSS      TXSTA, TRMT      ; Checks to see if the TRMT bit is empty or '1'
        GOTO      Trans_1          ; goes to Trans if TRMT is not '1'
        movlw     0xAE              ; ID number chosen here
        movwf     TXREG            ; 0x7F moved to TXREG to be transmitted
TenMs                                         ; creates a 10ms delay
        nop                       ; one cycle
        movlw     TenMsH           ; Initialize COUNT
        movwf     COUNTH
        movlw     TenMsL
        movwf     COUNTL
Ten_1                                         ; Inner loop
        decfsz   COUNTL,F          ; if COUNTL is not 0 then it goes to Ten_1
        goto     Ten_1
        movlw     TenMsL
        movwf     COUNTL
        decfsz   COUNTH,F          ; Outer loop
        goto     Ten_1             ; if COUNTH is not 0 then it goes to Ten_1
        Goto     Trans

; Feature=A2D - A2D configuration

; set pins for analog or digital

; B7:6=ADCS1:0 B5:3=CHS2:0 B2=GO B0=ADON
        clrf     ADCON0            ; GO bit2 to 0

; B7=ADFM B6=ADCS2 B3:0=PCFG3:0
        movlw   0x07
        movwf   ADCON1

; Feature=required - Interrupt flags cleared and interrupt configuration

; B7=RBPU-L B6:3=INTEDG0:3 B2=TMR0IP B1=INT3IP B0=RBIP
        movlw   0xFF
        movwf   INTCON2

; B7:6=INT2:1IP B5:3=INT3:1IE B2:0=INT3:1IF
        movlw   0xC0
        movwf   INTCON3

END

```

Appendix D: Spartan Code

Parallel_rx

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Parallel_rx is
  Port ( Serial_in : in STD_LOGIC;
        clk : in STD_LOGIC;
        reset: in STD_LOGIC;
        Parallel_out : out STD_LOGIC_VECTOR (19 downto 0));
end Parallel_rx;

architecture Behavioral of Parallel_rx is

begin

-- saves incoming data into a parallel output
process(clk, reset)
  variable count: integer range 0 to 31;
begin
  -- check if reset is pressed
  if (reset = '1') then
    count := 0;
    Parallel_out <= "00000000000000000000";
  else
    -- check if rising clock edge
    if (rising_edge(clk)) then
      Parallel_out(count) <= Serial_in;    -- saves data to parallel output
      count := count + 1;                -- increments count
      if( count = 20 ) then              -- after 20 bits taken reset count
        count := 0;
      end if;
    else
      count := count;
    end if;
  end if;
end process;

end Behavioral;
```

Appendix E: Spartan Code

Hex_Driver2w2

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Hex_Driver2w2 is
    Port ( num : in  STD_LOGIC_VECTOR (19 downto 0);
          good1 : out STD_LOGIC_VECTOR (7 downto 0);
          loadx : out STD_LOGIC;
          loady : out STD_LOGIC;
          loadz : out STD_LOGIC);
end Hex_Driver2w2;

architecture Behavioral of Hex_Driver2w2 is

    -- Variables
    signal loadxtemp : std_logic;
    signal loadytemp : std_logic;
    signal loadztemp : std_logic;
    signal good_temp : std_logic_vector(9 downto 0);

begin

    -- saves 10 bits when the sequence is detected
    good_temp <= num(9 downto 0) when (num(19 downto 9) = "10111111101")else
        "0000000000";

    -- increments an output to load data into memory
    process( num )
        variable count : integer range 0 to 4;
    begin
        if( num(19 downto 9) = "10111111101" ) then
            count := count + 1;
            if( count = 1) then
                loadztemp <= '0';
                loadxtemp <= '1';
                loadytemp <= '0';
            end if;
            if( count = 2) then
                loadztemp <= '0';
                loadxtemp <= '0';
            end if;
        end if;
    end process;
end Behavioral;
```

```

        loadytemp <= '1';
    end if;
    if( count = 3 ) then
    count := 0;
        loadztemp <= '1';
        loadxtemp <= '0';
        loadytemp <= '0';
    end if;
    else
        -- if the sequence is not detected then set to '0'
        loadztemp <= '0';
        loadxtemp <= '0';
        loadytemp <= '0';
    end if;
end process;

loadx <= loadxtemp;           -- stores loadxtemp to loadx to be outputted
loady <= loadytemp;           -- stores loadytemp to loady to be outputted
loadz <= loadztemp;           -- stores loadztemp to loadz to be outputted
good1(7 downto 0) <= good_temp(8 downto 1); -- takes only 8 bits of good_temp to be
outputted

end Behavioral;
```

Appendix F: Spartan Code

comp

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity comp is
    Port ( P      : in STD_LOGIC_VECTOR (7 downto 0);
          L      : in STD_LOGIC_VECTOR (7 downto 0);
          M      : in STD_LOGIC_VECTOR (7 downto 0);
          load    : in STD_LOGIC;
          rst     : in STD_LOGIC;
          clk24   : in STD_LOGIC;
          ID     : out STD_LOGIC_VECTOR (7 downto 0));
end comp;

architecture Behavioral of comp is

    -- Variables
    signal PL: std_logic_vector (7 downto 0);
    signal PM: std_logic_vector (7 downto 0);
    signal LM: std_logic_vector (7 downto 0);
    signal P_temp: std_logic_vector (7 downto 0);
    signal M_temp: std_logic_vector (7 downto 0);
    signal L_temp: std_logic_vector (7 downto 0);
    signal good: std_logic_vector (7 downto 0);

begin

    --hold values for a short time
    process(P, L, M)
        variable count : integer range 0 to 8;
    begin
        if (rst = '1') then
            count := 0;
            -- resets count if reset is pressed
        else
            if (load = '0') then
                -- if load is '0' then send 0d
                P_temp <= "00000000";
                L_temp <= "00000000";
                M_temp <= "00000000";
            else
                if(load = '1') then
                    -- if load is '1' then move inputs to a temp
                    variable

```

```

        if (rising_edge(clk24)) then
            P_temp(count) <= P(count);
            L_temp(count) <= L(count);
            M_temp(count) <= M(count);
            count := count + 1;           -- increments count
        if (count = 8) then             -- resets count after all bits moved
            count := 0;
        end if;
        else
            count := count;             -- keep count if not 8
        end if;
    end if;
end if;
end process;

--compare values saved in the temp variables
process(P_temp, L_temp, M_temp)
begin
    if (rst = '1') then                -- send 0d if reset is '1'
        good <="00000000";
    else
        if (load = '1') then
            if(P_temp(7 downto 0) = L_temp(7 downto 0)) then    -- compare two words
                PL (7 downto 0) <= P_temp(7 downto 0);          -- save P_temp to PL if equal
            else
                PL <= "00000000";                                -- if not equal then set PL to 0d
            end if;

            if(P_temp(7 downto 0) = M_temp(7 downto 0)) then    -- compare two words
                PM (7 downto 0) <= P_temp(7 downto 0);          -- save P_temp to PM if equal
            else
                PM <= "00000000";                                -- if not equal then set PM to 0d
            end if;

            if(L_temp(7 downto 0) = M_temp(7 downto 0)) then    -- compare two words
                LM (7 downto 0) <= M_temp(7 downto 0);          -- save M_temp to LM if equal
            else
                LM <= "00000000";                                -- if not equal then set LM to 0d
            end if;

            if (PL = PM) then                -- compare PL and PM
                good <= PL;                  -- if equal set PL to good
            else
                good <= "00000000";         -- otherwise good is 0d
            end if;
        end if;
    end if;
end process;

```

```
        end if;  
    end if;
```

```
end process;
```

```
ID (7 downto 0) <= good (7 downto 0);           -- set good to output ID  
end Behavioral;
```

Appendix G: Spartan Code

serial_tx2

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity serial_tx2 is
  port(
    data_in  : in std_logic_vector( 7 downto 0 );
    door_in  : in std_logic_vector( 7 downto 0 );
    tx       : in std_logic;
    clk      : in std_logic;
    rst      : in std_logic;
    finished : out std_logic;
    ser_out  : out std_logic);
end serial_tx2;
```

architecture Behavioral of serial_tx2 is

```
-- 1 start
-- 8 data
-- 1 parity (even)
-- 2 stop bits
signal tx_data : std_logic_vector( 23 downto 0 );

-- 9600 baud clk
signal clk_9600 : std_logic;

-- bit transmitted
signal ser_out_temp : std_logic;

-- continue
signal continue : std_logic;

-- transmission finished_temp
signal finished_temp : std_logic;

signal rst_temp : std_logic;
```

begin

```
-- start bit
tx_data(0) <= '0';
```

```

-- ID data
tx_data(8 downto 1) <= data_in (7 downto 0);

-- parity
tx_data(9) <= tx_data(1) XOR
    tx_data(2) XOR
    tx_data(3) XOR
    tx_data(4) XOR
    tx_data(5) XOR
    tx_data(6) XOR
    tx_data(7) XOR
    tx_data(8);

-- stop bits
tx_data(11 downto 10) <= "11";

-- start bit
tx_data(12) <= '0';

-- door data
tx_data(20 downto 13) <= door_in( 7 downto 0);

-- parity
tx_data(21) <= tx_data(13) XOR
    tx_data(14) XOR
    tx_data(15) XOR
    tx_data(16) XOR
    tx_data(17) XOR
    tx_data(18) XOR
    tx_data(19) XOR
    tx_data(20);

-- stop bits
tx_data(23 downto 22) <= "11";

-- 2400 baud clk
-- 50MHz/2400 = 20833
process( clk, rst )
    variable count : integer range 0 to 32768;
begin
    if( rising_edge(clk) ) then
        if( rst = '1' ) then
            clk_2400 <= '0';
            count := 0;
        else
            count := count + 1;
        end if;
    end if;
end process;

```

```

    if( count = 10416 ) then
        count := 0;
        clk_2400 <= not clk_2400;
    end if;
end if;
end if;
end process;

-- continue signal
process( clk, tx, finished_temp )
begin
    if( rising_edge(clk) ) then
        if( continue = '1' ) then
            continue <= continue;
        else
            if( tx = '1' ) then
                continue <= '1';
            else
                if ( finished_temp = '1' ) then
                    continue <= '0';
                else
                    continue <= continue;
                end if;
            end if;
        end if;
    end if;
end process;

-- transmit
process( clk_2400, rst, continue )
    variable count : integer range 0 to 30;
begin
    if( rst = '1' ) then
        count := 0;
        ser_out_temp <= '1';
        finished_temp <= '0';
    else
        if( continue = '1' ) then
            if( rising_edge( clk_2400 ) ) then
                ser_out_temp <= tx_data( count );
                count := count + 1;
                -- all bits transmitted?
                if( count = 24 ) then
                    finished_temp <= '1';
                    count := 0;
                    ser_out_temp <= '1';
                end if;
            end if;
        end if;
    end if;
end process;

```

```
    end if;
  else
    count := count;
    finished_temp <= finished_temp;
  end if;
end if;
end if;
end process;
```

```
ser_out <= ser_out_temp;
finished <= finished_temp;
```

```
end Behavioral;
```

Appendix H: Spartan Code

serial_tx2

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- This code generates five clock signals
-- (1 Hz, 10 Hz, 5 KHz, 9.6 KHz, 2.4 KHz) from a single
-- 50 MHz clock provided by the Spartan3 board

entity Clk_Convrtwa is
  Port ( Clk_in : in std_logic;
        Reset : in std_logic;
        Clk_1Hz, Clk_10Hz, Clk_5KHz, Clk_9_6KHz, Clk_2_4KHz : out std_logic
        );
end Clk_Convrtwa;

architecture Behavioral of Clk_Convrtwa is

  signal tmp_clk_1Hz : std_logic:= '0';
  signal tmp_clk_10Hz : std_logic:= '0';
  signal tmp_Clk_5KHz : std_logic:= '0';
  signal tmp_Clk_9_6KHz : std_logic:= '0';
  signal tmp_Clk_2_4KHz : std_logic:= '0';

begin
  Clk_1Hz <= tmp_clk_1Hz;
  Clk_10Hz <= tmp_clk_10Hz;
  Clk_5KHz <= tmp_Clk_5KHz;
  Clk_9_6KHz <= tmp_Clk_9_6KHz;
  Clk_2_4KHz <= tmp_Clk_2_4KHz;

  process(Reset, Clk_in)
    variable counter_1Hz: integer range 0 TO 25_000_000;
    variable counter_10Hz: integer range 0 TO 2_500_000;
    variable counter_9_6KHz: integer range 0 to 2_604;
    variable counter_5KHz: integer range 0 TO 5_000;
    variable counter_2_4KHz: integer range 0 TO 10_420;

  begin
    if Reset = '1' then
      counter_1Hz := 0;
      counter_10Hz := 0;

```

```

        counter_5KHz := 0;
        counter_9_6KHz := 0;
        counter_2_4KHz := 0;
    elsif Clk_in'event and Clk_in = '1' then
        counter_1Hz := counter_1Hz+1;
        counter_10Hz := counter_10Hz+1;
        counter_5KHz := counter_5KHz+1;
        counter_9_6KHz := counter_9_6KHz+1;
        counter_2_4KHz := counter_2_4KHz+1;

        if counter_1Hz = 25_000_000 then
tmp_clk_1Hz <= not tmp_clk_1Hz;
            counter_1Hz := 0;
        end if;
        if counter_10Hz = 2_500_000 then
tmp_clk_10Hz <= not tmp_clk_10Hz;
            counter_10Hz := 0;
        end if;
        if counter_5KHz = 5_000 then
tmp_Clk_5KHz <= not tmp_Clk_5KHz;
            counter_5KHz := 0;
        end if;
        if counter_9_6KHz = 2_604 then
tmp_Clk_9_6KHz <= not tmp_Clk_9_6KHz;
            counter_9_6KHz := 0;
        end if;
        if counter_2_4KHz = 10_420 then
tmp_Clk_2_4KHz <= not tmp_Clk_2_4KHz;
            counter_2_4KHz := 0;
        end if;
    end if;
end process;

end Behavioral;

```

Appendix I: Computer Code

Main.cpp

```
#include "stdafx.h"
#include "serialport.h"
#include "winerror.h"
#include "CSpreadSheet.h"
#include <iostream>
#include <sstream>
#include <windows.h>
#include "SN.h"
#include <afxdb.h>
using namespace std;
using namespace Saunternot;

void SaveSP(int number, int place);
void ReadPID(int idnumber, int doornumber);

int main()
{
    CSerialPort port;
    COMMTIMEOUTS timeouts;

    int id[10];           //stores id number from serial port
    int door[10];        //stores door number from serial port
    int i;                //variable used for counting
    int k = 1;           //constant to keep program running
    int idnum1;          //variable to store first comparison
    int idnum2;          //variable to store second comparison
    int idnum;           //variable to store comparison of two variable above
    int doornum;         //variable to store door[1] data

    //initialize arrays to a value known
    for(i=0; i<10; i++)
    {
        id[i]=0;
        door[i]=0;
    }

    //open comm port
    port.Open(1, 2400, CSerialPort::EvenParity, 8,
              CSerialPort::TwoStopBits,
              CSerialPort::NoFlowControl);

    while (!port.IsOpen())
        cout<<"port not open"<< "\n";

    //set the time outs
    port.SetTimeouts(timeouts);
}
```

```

while (k == k)
{
    for(i=1;i<=10;i++)
    {
        port.ClearReadBuffer();
        port.Read(&id[i-1], 1); //reads the serial port and places
the data //into the address
        myptr points to which is id[i]
        port.Read(&door[i-1], 1); //reads the serial port and places
the data //into the address
        myptr points to which is door[i]
        //cout<<id[i-1]<<"\n";
        //cout<<door[i-1]<<"\n";
        //SaveSP(id[i-1], i); // used for testing purposes ...
saves id array into excel spreadsheet
    }

    //compares 3 values of id array for consistency
    if(id[1] == id[2])
        idnum1 = id[1];
    if(id[1] == id[3])
        idnum2 = id[1];
    if(idnum1 == idnum2)
        idnum = idnum1;

    doornum=door[1]; //sets first position of door as a
variable called doornum

    ReadPID(idnum, doornum); //sends idnum and doornum to
ReadPID
}

port.Close();

return 0;
}

```

//SaveSP takes in a number and a place holder and returns nothing

```

void SaveSP(int number, int place)
{
    CSpreadSheet SS("Reader.xls", "ID");
    CStringArray sampleArray, testRow, Rows, Column;
    CString tempString;
    int i;
    int j;
    int k;
    int num[1000];

    k=place-1;

```

```

    if (k==0)
    {
        sampleArray.RemoveAll();
        for(k=0; k<1000; k++)
        {
            num[k]=0;
        }
    }

    num[k]=number;

    // saves data from reader to excel
    SS.BeginTransaction();

    tempString.Format("%d", num[k]);
    sampleArray.Add(tempString);
    SS.AddRow(sampleArray, place, true);

    SS.Commit();

    SS.Convert(","); // convert Excel spreadsheet into text delimited
format                                     // with , as separator

    return;
}

//ReadPID takes in an idnumber and a doornumber and returns nothing
void ReadPID(int idnumber, int doornumber)
{
    CSpreadSheet SNID("SNID.xls", "SNID");
    CStringArray sampleArray, testRow, Rows, Column;
    CString tempString, ID, DOOR;
    Saunternot::SN snForm;
    System::Windows::Forms::DialogResult result =
snForm.DialogResult::get();
    // Create a new Excel spreadsheet, filename is test.xls, sheetname is
TestSheet

    int i;
    int j;
    int k;

    ID.Format("%d.0", idnumber);
    DOOR.Format("%d", doornumber);

    // read data from SNID.xls
    SNID.BeginTransaction();

    SNID.Convert(","); // convert Excel spreadsheet into text delimited
format                                     // with ; as separator

    // Read and print out contents of second column of spreadsheet
    SNID.ReadColumn(Column, 1);

```

```

for (i = 1; i <= Column.GetSize(); i++)
{
    SNID.ReadCell(tempString, 1, i);
    //cout<<tempString<<"\n";

    if(tempString == ID)
    {
        SNID.ReadRow(Rows, i);
        SNID.ReadCell(tempString, 2, i);
        //printf("\nCell value at (%d,2): %s\n", idnumber,
tempString);

        if (tempString != "")
        {

            // Convert to a System::String
            String ^systemstring = gcnew String(tempString);
            String ^doorstring = gcnew String (DOOR);

            snForm.labell->Text = systemstring += " has exited
the facility using door " + doorstring + ".";
            snForm.Enabled = true;
            snForm.Visible = false;
            snForm.Activate();
            Beep(1000, 1000);
            snForm.ShowDialog();

        }
    }
}

return;
}

```

Appendix J: Computer Code

serialport.h

```
/*
Module : SERIALPORT.H
Purpose: Declaration for an MFC wrapper class for serial ports
Created: PJN / 31-05-1999
History: None

Copyright (c) 1999 by PJ Naughter.
All rights reserved.

*/

////////////////////////////////// Macros / Structs etc //////////////////////////////////

#ifndef __SERIALPORT_H__
#define __SERIALPORT_H__

////////////////////////////////// Classes //////////////////////////////////

//////// Serial port exception class
//////////////////////////////////

void AfxThrowSerialException(DWORD dwError = 0);

class CSerialException : public CException
{
public:
//Constructors / Destructors
    CSerialException(DWORD dwError);
    ~CSerialException();

//Methods
#ifdef _DEBUG
    virtual void Dump(CDumpContext& dc) const;
#endif
    virtual BOOL GetErrorMessage(LPTSTR lpstrError, UINT nMaxError, PUINT
pnHelpContext = NULL);
    CString GetErrorMessage();

//Data members
    DWORD m_dwError;

protected:
    DECLARE_DYNAMIC(CSerialException)
};
```

```

//// The actual serial port class
////////////////////////////////////

class CSerialPort : public CObject
{
public:
//Enums
enum FlowControl
{
    NoFlowControl,
    CtsRtsFlowControl,
    CtsDtrFlowControl,
    DsrRtsFlowControl,
    DsrDtrFlowControl,
    XonXoffFlowControl
};

enum Parity
{
    EvenParity,
    MarkParity,
    NoParity,
    OddParity,
    SpaceParity
};

enum StopBits
{
    OneStopBit,
    OnePointFiveStopBits,
    TwoStopBits
};

//Constructors / Destructors
CSerialPort();
~CSerialPort();

//General Methods
void Open(int nPort, DWORD dwBaud = 9600, Parity parity = NoParity, BYTE
DataBits = 8,
        StopBits stopbits = OneStopBit, FlowControl fc = NoFlowControl,
        BOOL bOverlapped = FALSE);
void Close();
void Attach(HANDLE hComm);
HANDLE Detach();
operator HANDLE() const { return m_hComm; };
BOOL IsOpen() const { return m_hComm != INVALID_HANDLE_VALUE; };
#ifdef _DEBUG
void CSerialPort::Dump(CDumpContext& dc) const;
#endif

//Reading / Writing Methods
DWORD Read(void* lpBuf, DWORD dwCount);
BOOL Read(void* lpBuf, DWORD dwCount, OVERLAPPED& overlapped);
void ReadEx(void* lpBuf, DWORD dwCount);
DWORD Write(const void* lpBuf, DWORD dwCount);
BOOL Write(const void* lpBuf, DWORD dwCount, OVERLAPPED& overlapped);

```

```

void WriteEx(const void* lpBuf, DWORD dwCount);
void TransmitChar(char cChar);
void GetOverlappedResult(OVERLAPPED& overlapped, DWORD& dwBytesTransferred,
BOOL bWait);
void CancelIo();

//Configuration Methods
void GetConfig(COMMCONFIG& config);
static void GetDefaultConfig(int nPort, COMMCONFIG& config);
void SetConfig(COMMCONFIG& Config);
static void SetDefaultConfig(int nPort, COMMCONFIG& config);

//Misc RS232 Methods
void ClearBreak();
void SetBreak();
void ClearError(DWORD& dwErrors);
void GetStatus(COMSTAT& stat);
void GetState(DCB& dcb);
void SetState(DCB& dcb);
void Escape(DWORD dwFunc);
void ClearDTR();
void ClearRTS();
void SetDTR();
void SetRTS();
void SetXOFF();
void SetXON();
void GetProperties(COMMPROP& properties);
void GetModemStatus(DWORD& dwModemStatus);

//Timeouts
void SetTimeouts(COMMTIMEOUTS& timeouts);
void GetTimeouts(COMMTIMEOUTS& timeouts);
void Set0Timeout();
void Set0WriteTimeout();
void Set0ReadTimeout();

//Event Methods
void SetMask(DWORD dwMask);
void GetMask(DWORD& dwMask);
void WaitEvent(DWORD& dwMask);
void WaitEvent(DWORD& dwMask, OVERLAPPED& overlapped);

//Queue Methods
void Flush();
void Purge(DWORD dwFlags);
void TerminateOutstandingWrites();
void TerminateOutstandingReads();
void ClearWriteBuffer();
void ClearReadBuffer();
void Setup(DWORD dwInQueue, DWORD dwOutQueue);

//Overridables
virtual void OnCompletion(DWORD dwErrorCode, DWORD dwCount, LPOVERLAPPED
lpOverlapped);

protected:
HANDLE m_hComm;          //Handle to the comms port

```

```

    BOOL    m_bOverlapped; //Is the port open in overlapped IO

    static void WINAPI _OnCompletion(DWORD dwErrorCode, DWORD dwCount,
LPOVERLAPPED lpOverlapped);

        DECLARE_DYNAMIC(CSerialPort)
};

#endif //__SERIALPORT_H__

```

serialport.cpp

```

/*
Module : SERIALPORT.CPP
Purpose: Implementation for an MFC wrapper class for serial ports
Created: PJN / 31-05-1999
History: PJN / 03-06-1999 1. Fixed problem with code using CancelIo which
does not exist on 95.
                                2. Fixed leaks which can occur in sample app when
an exception is thrown
        PJN / 16-06-1999 1. Fixed a bug whereby CString::ReleaseBuffer was
not being called in
                                CStringException::GetErrorMessage
        PJN / 29-09-1999 1. Fixed a simple copy and paste bug in
CSerialPort::SetDTR

Copyright (c) 1999 by PJ Naughter.
All rights reserved.

*/

//////////////////////////////////// Includes
////////////////////////////////////
#include "stdafx.h"
#include "serialport.h"
#include "winerror.h"

//////////////////////////////////// defines
////////////////////////////////////

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////// Implementation
////////////////////////////////////

//Class which handles CancelIo function which must be constructed at run time
//since it is not implemented on NT 3.51 or Windows 95. To avoid the loader
//bringing up a message such as "Failed to load due to missing export...",
the

```

```

//function is constructed using GetProcAddress. The CSerialPort::CancelIo
//function then checks to see if the function pointer is NULL and if it is it
//throws an exception using the error code ERROR_CALL_NOT_IMPLEMENTED which
//is what 95 would have done if it had implemented a stub for it in the first
//place !!

```

```

class _SERIAL_PORT_DATA
{
public:
//Constructors /Destructors
    _SERIAL_PORT_DATA();
    ~_SERIAL_PORT_DATA();

    HINSTANCE m_hKernel32;
    typedef BOOL (CANCELIO) (HANDLE);
    typedef CANCELIO* LPCANCELIO;
    LPCANCELIO m_lpfncancelIo;
};

_SERIAL_PORT_DATA::_SERIAL_PORT_DATA()
{
    m_hKernel32 = LoadLibrary(_T("KERNEL32.DLL"));
    VERIFY(m_hKernel32 != NULL);
    m_lpfncancelIo = (LPCANCELIO) GetProcAddress(m_hKernel32, "CancelIo");
}

_SERIAL_PORT_DATA::~~_SERIAL_PORT_DATA()
{
    FreeLibrary(m_hKernel32);
    m_hKernel32 = NULL;
}

//The local variable which handle the function pointers
_SERIAL_PORT_DATA _SerialPortData;

////////// Exception handling code

void AfxThrowSerialException(DWORD dwError /* = 0 */)
{
    if (dwError == 0)
        dwError = ::GetLastError();

    CSerialException* pException = new CSerialException(dwError);

    TRACE(_T("Warning: throwing CSerialException for error %d\n"),
dwError);
    THROW(pException);
}

BOOL CSerialException::GetErrorMessage(LPTSTR pstrError, UINT nMaxError,
PUINT pnHelpContext)
{
    ASSERT(pstrError != NULL && AfxIsValidString(pstrError, nMaxError));
}

```

```

        if (pnHelpContext != NULL)
            *pnHelpContext = 0;

        LPTSTR lpBuffer;
        BOOL bRet = FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
FORMAT_MESSAGE_FROM_SYSTEM,
                                NULL, m_dwError,
MAKELANGID(LANG_NEUTRAL, SUBLANG_SYS_DEFAULT),
                                (LPTSTR) &lpBuffer, 0, NULL);

        if (bRet == FALSE)
            *pstrError = '\\0';
        else
        {
            lstrcpyn(pstrError, lpBuffer, nMaxError);
            bRet = TRUE;

            LocalFree(lpBuffer);
        }

        return bRet;
    }

CString CSerialException::GetErrorMessage()
{
    CString rval;
    LPTSTR pstrError = rval.GetBuffer(4096);
    GetErrorMessage(pstrError, 4096, NULL);
    rval.ReleaseBuffer();
    return rval;
}

CSerialException::CSerialException(DWORD dwError)
{
    m_dwError = dwError;
}

CSerialException::~CSerialException()
{
}

IMPLEMENT_DYNAMIC(CSerialException, CException)

#ifdef _DEBUG
void CSerialException::Dump(CDumpContext& dc) const
{
    CObject::Dump(dc);

    dc << "m_dwError = " << m_dwError;
}
#endif

////////// The actual serial port code

CSerialPort::CSerialPort()
{
    m_hComm = INVALID_HANDLE_VALUE;
}

```

```

    m_bOverlapped = FALSE;
}

CSerialPort::~CSerialPort()
{
    Close();
}

IMPLEMENT_DYNAMIC(CSerialPort, CObject)

#ifdef _DEBUG
void CSerialPort::Dump(CDumpContext& dc) const
{
    CObject::Dump(dc);

    dc << _T("m_hComm = ") << m_hComm << _T("\n");
    dc << _T("m_bOverlapped = ") << m_bOverlapped;
}
#endif

void CSerialPort::Open(int nPort, DWORD dwBaud, Parity parity, BYTE DataBits,
StopBits stopbits, FlowControl fc, BOOL bOverlapped)
{
    //Validate our parameters
    ASSERT(nPort>0 && nPort<=255);

    //Call CreateFile to open up the comms port
    CString sPort;
    sPort.Format(_T("\\\\.\\COM%d"), nPort);
    m_hComm = CreateFile(sPort, GENERIC_READ | GENERIC_WRITE, 0, NULL,
OPEN_EXISTING, bOverlapped ? FILE_FLAG_OVERLAPPED : 0, NULL);
    if (m_hComm == INVALID_HANDLE_VALUE)
    {
        TRACE(_T("Failed to open up the comms port\n"));
        AfxThrowSerialException();
    }

    m_bOverlapped = bOverlapped;

    //Get the current state prior to changing it
    DCB dcb;
    GetState(dcb);

    //Setup the baud rate
    dcb.BaudRate = dwBaud;

    //Setup the Parity
    switch (parity)
    {
        case EvenParity:    dcb.Parity = EVENPARITY;    break;
        case MarkParity:    dcb.Parity = MARKPARITY;    break;
        case NoParity:      dcb.Parity = NOPARITY;      break;
        case OddParity:     dcb.Parity = ODDPARITY;     break;
        case SpaceParity:   dcb.Parity = SPACEPARITY;   break;
        default:            ASSERT(FALSE);              break;
    }
}

```

```

//Setup the data bits
dcb.ByteSize = DataBits;

//Setup the stop bits
switch (stopbits)
{
    case OneStopBit:          dcb.StopBits = ONESTOPBIT;    break;
    case OnePointFiveStopBits: dcb.StopBits = ONE5STOPBITS; break;
    case TwoStopBits:         dcb.StopBits = TWOSTOPBITS;   break;
    default:                  ASSERT(FALSE);                break;
}

//Setup the flow control
dcb.fDsrSensitivity = FALSE;
switch (fc)
{
    case NoFlowControl:
    {
        dcb.fOutxCtsFlow = FALSE;
        dcb.fOutxDsrFlow = FALSE;
        dcb.fOutX = FALSE;
        dcb.fInX = FALSE;
        break;
    }
    case CtsRtsFlowControl:
    {
        dcb.fOutxCtsFlow = TRUE;
        dcb.fOutxDsrFlow = FALSE;
        dcb.fRtsControl = RTS_CONTROL_HANDSHAKE;
        dcb.fOutX = FALSE;
        dcb.fInX = FALSE;
        break;
    }
    case CtsDtrFlowControl:
    {
        dcb.fOutxCtsFlow = TRUE;
        dcb.fOutxDsrFlow = FALSE;
        dcb.fDtrControl = DTR_CONTROL_HANDSHAKE;
        dcb.fOutX = FALSE;
        dcb.fInX = FALSE;
        break;
    }
    case DsrRtsFlowControl:
    {
        dcb.fOutxCtsFlow = FALSE;
        dcb.fOutxDsrFlow = TRUE;
        dcb.fRtsControl = RTS_CONTROL_HANDSHAKE;
        dcb.fOutX = FALSE;
        dcb.fInX = FALSE;
        break;
    }
    case DsrDtrFlowControl:
    {
        dcb.fOutxCtsFlow = FALSE;
        dcb.fOutxDsrFlow = TRUE;
        dcb.fDtrControl = DTR_CONTROL_HANDSHAKE;
        dcb.fOutX = FALSE;
    }
}

```

```

        dcb.fInX = FALSE;
        break;
    }
    case XonXoffFlowControl:
    {
        dcb.fOutxCtsFlow = FALSE;
        dcb.fOutxDsrFlow = FALSE;
        dcb.fOutX = TRUE;
        dcb.fInX = TRUE;
        dcb.XonChar = 0x11;
        dcb.XoffChar = 0x13;
        dcb.XoffLim = 100;
        dcb.XonLim = 100;
        break;
    }
    default:
    {
        ASSERT(FALSE);
        break;
    }
}

//Now that we have all the settings in place, make the changes
SetState(dcb);
}

void CSerialPort::Close()
{
    if (IsOpen())
    {
        BOOL bSuccess = CloseHandle(m_hComm);
        m_hComm = INVALID_HANDLE_VALUE;
        if (!bSuccess)
            TRACE(_T("Failed to close up the comms port, GetLastError:%d\n"),
                GetLastError());
        m_bOverlapped = FALSE;
    }
}

void CSerialPort::Attach(HANDLE hComm)
{
    Close();
    m_hComm = hComm;
}

HANDLE CSerialPort::Detach()
{
    HANDLE hrVal = m_hComm;
    m_hComm = INVALID_HANDLE_VALUE;
    return hrVal;
}

DWORD CSerialPort::Read(void* lpBuf, DWORD dwCount)
{
    ASSERT(IsOpen());
    ASSERT(!m_bOverlapped);
}

```

```

DWORD dwBytesRead = 0;
if (!ReadFile(m_hComm, lpBuf, dwCount, &dwBytesRead, NULL))
{
    TRACE(_T("Failed in call to ReadFile\n"));
    AfxThrowSerialException();
}

return dwBytesRead;
}

BOOL CSerialPort::Read(void* lpBuf, DWORD dwCount, OVERLAPPED& overlapped)
{
    ASSERT(IsOpen());
    ASSERT(m_bOverlapped);
    ASSERT(overlapped.hEvent);

    DWORD dwBytesRead = 0;
    BOOL bSuccess = ReadFile(m_hComm, lpBuf, dwCount, &dwBytesRead,
&overlapped);
    if (!bSuccess)
    {
        if (GetLastError() != ERROR_IO_PENDING)
        {
            TRACE(_T("Failed in call to ReadFile\n"));
            AfxThrowSerialException();
        }
    }
    return bSuccess;
}

DWORD CSerialPort::Write(const void* lpBuf, DWORD dwCount)
{
    ASSERT(IsOpen());
    ASSERT(!m_bOverlapped);

    DWORD dwBytesWritten = 0;
    if (!WriteFile(m_hComm, lpBuf, dwCount, &dwBytesWritten, NULL))
    {
        TRACE(_T("Failed in call to WriteFile\n"));
        AfxThrowSerialException();
    }

    return dwBytesWritten;
}

BOOL CSerialPort::Write(const void* lpBuf, DWORD dwCount, OVERLAPPED&
overlapped)
{
    ASSERT(IsOpen());
    ASSERT(m_bOverlapped);
    ASSERT(overlapped.hEvent);

    DWORD dwBytesWritten = 0;
    BOOL bSuccess = WriteFile(m_hComm, lpBuf, dwCount, &dwBytesWritten,
&overlapped);
    if (!bSuccess)
    {

```

```

        if (GetLastError() != ERROR_IO_PENDING)
        {
            TRACE(_T("Failed in call to WriteFile\n"));
            AfxThrowSerialException();
        }
    }
    return bSuccess;
}

void CSerialPort::GetOverlappedResult(OVERLAPPED& overlapped, DWORD&
dwBytesTransferred, BOOL bWait)
{
    ASSERT(IsOpen());
    ASSERT(m_bOverlapped);
    ASSERT(overlapped.hEvent);

    DWORD dwBytesWritten = 0;
    if (!::GetOverlappedResult(m_hComm, &overlapped, &dwBytesTransferred,
bWait))
    {
        if (GetLastError() != ERROR_IO_PENDING)
        {
            TRACE(_T("Failed in call to GetOverlappedResult\n"));
            AfxThrowSerialException();
        }
    }
}

void CSerialPort::_OnCompletion(DWORD dwErrorCode, DWORD dwCount,
LPOVERLAPPED lpOverlapped)
{
    //Validate our parameters
    ASSERT(lpOverlapped);

    //Convert back to the C++ world
    CSerialPort* pSerialPort = (CSerialPort*) lpOverlapped->hEvent;
    ASSERT(pSerialPort->IsKindOf(RUNTIME_CLASS(CSerialPort)));

    //Call the C++ function
    pSerialPort->OnCompletion(dwErrorCode, dwCount, lpOverlapped);
}

void CSerialPort::OnCompletion(DWORD /*dwErrorCode*/, DWORD /*dwCount*/,
LPOVERLAPPED lpOverlapped)
{
    //Just free up the memory which was previously allocated for the OVERLAPPED
structure
    delete lpOverlapped;

    //Your derived classes can do something useful in OnCompletion, but don't
forget to
    //call CSerialPort::OnCompletion to ensure the memory is freed up
}

void CSerialPort::CancelIo()
{
    ASSERT(IsOpen());
}

```

```

    if (_SerialPortData.m_lpfncancelIo == NULL)
    {
        TRACE(_T("CancelIo function is not supported on this OS. You need to be
running at least NT 4 or Win 98 to use this function\n"));
        AfxThrowSerialException(ERROR_CALL_NOT_IMPLEMENTED);
    }

    if (!::_SerialPortData.m_lpfncancelIo(m_hComm))
    {
        TRACE(_T("Failed in call to CancelIO\n"));
        AfxThrowSerialException();
    }
}

void CSerialPort::WriteEx(const void* lpBuf, DWORD dwCount)
{
    ASSERT(IsOpen());

    OVERLAPPED* pOverlapped = new OVERLAPPED;
    ZeroMemory(pOverlapped, sizeof(OVERLAPPED));
    pOverlapped->hEvent = (HANDLE) this;
    if (!WriteFileEx(m_hComm, lpBuf, dwCount, pOverlapped, _OnCompletion))
    {
        delete pOverlapped;
        TRACE(_T("Failed in call to WriteFileEx\n"));
        AfxThrowSerialException();
    }
}

void CSerialPort::ReadEx(void* lpBuf, DWORD dwCount)
{
    ASSERT(IsOpen());

    OVERLAPPED* pOverlapped = new OVERLAPPED;
    ZeroMemory(pOverlapped, sizeof(OVERLAPPED));
    pOverlapped->hEvent = (HANDLE) this;
    if (!ReadFileEx(m_hComm, lpBuf, dwCount, pOverlapped, _OnCompletion))
    {
        delete pOverlapped;
        TRACE(_T("Failed in call to ReadFileEx\n"));
        AfxThrowSerialException();
    }
}

void CSerialPort::TransmitChar(char cChar)
{
    ASSERT(IsOpen());

    if (!TransmitCommChar(m_hComm, cChar))
    {
        TRACE(_T("Failed in call to TransmitCommChar\n"));
        AfxThrowSerialException();
    }
}

void CSerialPort::GetConfig(COMMCONFIG& config)

```

```

{
    ASSERT(IsOpen());

    DWORD dwSize = sizeof(COMMCONFIG);
    if (!GetCommConfig(m_hComm, &config, &dwSize))
    {
        TRACE(_T("Failed in call to GetCommConfig\n"));
        AfxThrowSerialException();
    }
}

void CSerialPort::SetConfig(COMMCONFIG& config)
{
    ASSERT(IsOpen());

    DWORD dwSize = sizeof(COMMCONFIG);
    if (!SetCommConfig(m_hComm, &config, dwSize))
    {
        TRACE(_T("Failed in call to SetCommConfig\n"));
        AfxThrowSerialException();
    }
}

void CSerialPort::SetBreak()
{
    ASSERT(IsOpen());

    if (!SetCommBreak(m_hComm))
    {
        TRACE(_T("Failed in call to SetCommBreak\n"));
        AfxThrowSerialException();
    }
}

void CSerialPort::ClearBreak()
{
    ASSERT(IsOpen());

    if (!ClearCommBreak(m_hComm))
    {
        TRACE(_T("Failed in call to SetCommBreak\n"));
        AfxThrowSerialException();
    }
}

void CSerialPort::ClearError(DWORD& dwErrors)
{
    ASSERT(IsOpen());

    if (!ClearCommError(m_hComm, &dwErrors, NULL))
    {
        TRACE(_T("Failed in call to ClearCommError\n"));
        AfxThrowSerialException();
    }
}

void CSerialPort::GetDefaultConfig(int nPort, COMMCONFIG& config)

```

```

{
    //Validate our parameters
    ASSERT(nPort>0 && nPort<=255);

    //Create the device name as a string
    CString sPort;
    sPort.Format(_T("COM%d"), nPort);

    DWORD dwSize = sizeof(COMMCONFIG);
    if (!GetDefaultCommConfig(sPort, &config, &dwSize))
    {
        TRACE(_T("Failed in call to GetDefaultCommConfig\n"));
        AfxThrowSerialException();
    }
}

void CSerialPort::SetDefaultConfig(int nPort, COMMCONFIG& config)
{
    //Validate our parameters
    ASSERT(nPort>0 && nPort<=255);

    //Create the device name as a string
    CString sPort;
    sPort.Format(_T("COM%d"), nPort);

    DWORD dwSize = sizeof(COMMCONFIG);
    if (!SetDefaultCommConfig(sPort, &config, dwSize))
    {
        TRACE(_T("Failed in call to GetDefaultCommConfig\n"));
        AfxThrowSerialException();
    }
}

void CSerialPort::GetStatus(COMSTAT& stat)
{
    ASSERT(IsOpen());

    DWORD dwErrors;
    if (!ClearCommError(m_hComm, &dwErrors, &stat))
    {
        TRACE(_T("Failed in call to ClearCommError\n"));
        AfxThrowSerialException();
    }
}

void CSerialPort::GetState(DCB& dcb)
{
    ASSERT(IsOpen());

    if (!GetCommState(m_hComm, &dcb))
    {
        TRACE(_T("Failed in call to GetCommState\n"));
        AfxThrowSerialException();
    }
}

void CSerialPort::SetState(DCB& dcb)

```

```

{
    ASSERT(IsOpen());

    if (!SetCommState(m_hComm, &dcb))
    {
        TRACE(_T("Failed in call to SetCommState\n"));
        AfxThrowSerialException();
    }
}

void CSerialPort::Escape(DWORD dwFunc)
{
    ASSERT(IsOpen());

    if (!EscapeCommFunction(m_hComm, dwFunc))
    {
        TRACE(_T("Failed in call to EscapeCommFunction\n"));
        AfxThrowSerialException();
    }
}

void CSerialPort::ClearDTR()
{
    Escape(CLRDTR);
}

void CSerialPort::ClearRTS()
{
    Escape(CLRRTS);
}

void CSerialPort::SetDTR()
{
    Escape(SETDTR);
}

void CSerialPort::SetRTS()
{
    Escape(SETRTS);
}

void CSerialPort::SetXOFF()
{
    Escape(SETXOFF);
}

void CSerialPort::SetXON()
{
    Escape(SETXON);
}

void CSerialPort::GetProperties(COMMPROP& properties)
{
    ASSERT(IsOpen());

    if (!GetCommProperties(m_hComm, &properties))
    {

```

```

        TRACE(_T("Failed in call to GetCommProperties\n"));
        AfxThrowSerialException();
    }
}

void CSerialPort::GetModemStatus(DWORD& dwModemStatus)
{
    ASSERT(IsOpen());

    if (!GetCommModemStatus(m_hComm, &dwModemStatus))
    {
        TRACE(_T("Failed in call to GetCommModemStatus\n"));
        AfxThrowSerialException();
    }
}

void CSerialPort::SetMask(DWORD dwMask)
{
    ASSERT(IsOpen());

    if (!SetCommMask(m_hComm, dwMask))
    {
        TRACE(_T("Failed in call to SetCommMask\n"));
        AfxThrowSerialException();
    }
}

void CSerialPort::GetMask(DWORD& dwMask)
{
    ASSERT(IsOpen());

    if (!GetCommMask(m_hComm, &dwMask))
    {
        TRACE(_T("Failed in call to GetCommMask\n"));
        AfxThrowSerialException();
    }
}

void CSerialPort::Flush()
{
    ASSERT(IsOpen());

    if (!FlushFileBuffers(m_hComm))
    {
        TRACE(_T("Failed in call to FlushFileBuffers\n"));
        AfxThrowSerialException();
    }
}

void CSerialPort::Purge(DWORD dwFlags)
{
    ASSERT(IsOpen());

    if (!PurgeComm(m_hComm, dwFlags))
    {
        TRACE(_T("Failed in call to PurgeComm\n"));
        AfxThrowSerialException();
    }
}

```

```

    }
}

void CSerialPort::TerminateOutstandingWrites()
{
    Purge(PURGE_TXABORT);
}

void CSerialPort::TerminateOutstandingReads()
{
    Purge(PURGE_RXABORT);
}

void CSerialPort::ClearWriteBuffer()
{
    Purge(PURGE_TXCLEAR);
}

void CSerialPort::ClearReadBuffer()
{
    Purge(PURGE_RXCLEAR);
}

void CSerialPort::Setup(DWORD dwInQueue, DWORD dwOutQueue)
{
    ASSERT(IsOpen());

    if (!SetupComm(m_hComm, dwInQueue, dwOutQueue))
    {
        TRACE(_T("Failed in call to SetupComm\n"));
        AfxThrowSerialException();
    }
}

void CSerialPort::SetTimeouts(COMMTIMEOUTS& timeouts)
{
    ASSERT(IsOpen());

    if (!SetCommTimeouts(m_hComm, &timeouts))
    {
        TRACE(_T("Failed in call to SetCommTimeouts\n"));
        AfxThrowSerialException();
    }
}

void CSerialPort::GetTimeouts(COMMTIMEOUTS& timeouts)
{
    ASSERT(IsOpen());

    if (!GetCommTimeouts(m_hComm, &timeouts))
    {
        TRACE(_T("Failed in call to GetCommTimeouts\n"));
        AfxThrowSerialException();
    }
}

void CSerialPort::Set0Timeout()

```

```

{
    COMMTIMEOUTS Timeouts;
    ZeroMemory(&Timeouts, sizeof(COMMTIMEOUTS));
    Timeouts.ReadIntervalTimeout = MAXDWORD;
    Timeouts.ReadTotalTimeoutMultiplier = 0;
    Timeouts.ReadTotalTimeoutConstant = 0;
    Timeouts.WriteTotalTimeoutMultiplier = 0;
    Timeouts.WriteTotalTimeoutConstant = 0;
    SetTimeouts(Timeouts);
}

void CSerialPort::Set0WriteTimeout()
{
    COMMTIMEOUTS Timeouts;
    GetTimeouts(Timeouts);
    Timeouts.WriteTotalTimeoutMultiplier = 0;
    Timeouts.WriteTotalTimeoutConstant = 0;
    SetTimeouts(Timeouts);
}

void CSerialPort::Set0ReadTimeout()
{
    COMMTIMEOUTS Timeouts;
    GetTimeouts(Timeouts);
    Timeouts.ReadIntervalTimeout = MAXDWORD;
    Timeouts.ReadTotalTimeoutMultiplier = 0;
    Timeouts.ReadTotalTimeoutConstant = 0;
    SetTimeouts(Timeouts);
}

void CSerialPort::WaitEvent(DWORD& dwMask)
{
    ASSERT(IsOpen());
    ASSERT(!m_bOverlapped);

    if (!WaitCommEvent(m_hComm, &dwMask, NULL))
    {
        TRACE(_T("Failed in call to WaitCommEvent\n"));
        AfxThrowSerialException();
    }
}

void CSerialPort::WaitEvent(DWORD& dwMask, OVERLAPPED& overlapped)
{
    ASSERT(IsOpen());
    ASSERT(m_bOverlapped);
    ASSERT(overlapped.hEvent);

    if (!WaitCommEvent(m_hComm, &dwMask, &overlapped))
    {
        if (GetLastError() != ERROR_IO_PENDING)
        {
            TRACE(_T("Failed in call to WaitCommEvent\n"));
            AfxThrowSerialException();
        }
    }
}

```

Appendix K: Computer Code

CSpreadSheet.h

```
// Class to read and write to Excel and text delimited spreadsheet
//
// Created by Yap Chun Wei
// December 2001
//
// Version 1.1
// Updates: Fix bug in ReadRow() which prevent reading of single column
spreadsheet

#ifndef CSPREADSHEET_H
#define CSPREADSHEET_H

#include <odbcinst.h>
#include <afxdb.h>

class CSpreadSheet
{
public:
    CSpreadSheet(CString File, CString SheetOrSeparator, bool Backup =
true); // Open spreadsheet for reading and writing
    ~CSpreadSheet(); // Perform some cleanup functions
    bool AddHeaders(CStringArray &FieldNames, bool replace = false); // Add
header row to spreadsheet
    bool DeleteSheet(); // Clear text delimited file content
    bool DeleteSheet(CString SheetName); // Clear entire Excel spreadsheet
content. The sheet itself is not deleted
    bool AddRow(CStringArray &RowValues, long row = 0, bool replace =
false); // Insert or replace a row into spreadsheet. Default is add new row.
    bool AddCell(CString CellValue, CString column, long row = 0, bool Auto
= true); // Replace or add a cell into Excel spreadsheet using header row or
column alphabet. Default is add cell into new row. Set Auto to false if want
to force column to be used as header name
    bool AddCell(CString CellValue, short column, long row = 0); // Replace
or add a cell into spreadsheet using column number. Default is add cell into
new row.
    bool ReplaceRows(CStringArray &NewRowValues, CStringArray
&OldRowValues); // Search and replace rows in Excel spreadsheet
    bool ReadRow(CStringArray &RowValues, long row = 0); // Read a row from
spreadsheet. Default is read the next row
    bool ReadColumn(CStringArray &ColumnValues, CString column, bool Auto =
true); // Read a column from Excel spreadsheet using header row or column
alphabet. Set Auto to false if want to force column to be used as header name
    bool ReadColumn(CStringArray &ColumnValues, short column); // Read a
column from spreadsheet using column number
    bool ReadCell (CString &CellValue, CString column, long row = 0, bool
Auto = true); // Read a cell from Excel spreadsheet using header row or
column alphabet. Default is read the next cell in next row. Set Auto to false
if want to force column to be used as header name
```

```

    bool ReadCell (CString &CellValue, short column, long row = 0); // Read
a cell from spreadsheet using column number. Default is read the next cell in
next row.
    void BeginTransaction(); // Begin transaction
    bool Commit(); // Save changes to spreadsheet
    bool RollBack(); // Undo changes to spreadsheet
    bool Convert(CString SheetOrSeparator);
    inline void GetFieldNames (CStringArray &FieldNames)
{FieldNames.RemoveAll(); FieldNames.Copy(m_aFieldNames);} // Get the header
row from spreadsheet
    inline long GetTotalRows() {return m_dTotalRows;} // Get total number
of rows in spreadsheet
    inline short GetTotalColumns() {return m_dTotalColumns;} // Get total
number of columns in spreadsheet
    inline long GetCurrentRow() {return m_dCurrentRow;} // Get the
currently selected row in spreadsheet
    inline bool GetBackupStatus() {return m_bBackup;} // Get status of
backup. True if backup is successful, False if spreadsheet is not backup
    inline bool GetTransactionStatus() {return m_bTransaction;} // Get
status of Transaction. True if Transaction is started, False if Transaction
is not started or has error in starting
    inline CString GetLastError() {return m_sLastError;} // Get last error
message

private:
    bool Open(); // Open a text delimited file for reading or writing
    void GetExcelDriver(); // Get the name of the Excel-ODBC driver
    short CalculateColumnNumber(CString column, bool Auto); // Convert
Excel column in alphabet into column number

    bool m_bAppend; // Internal flag to denote newly created spreadsheet or
previously created spreadsheet
    bool m_bBackup; // Internal flag to denote status of Backup
    bool m_bExcel; // Internal flag to denote whether file is Excel
spreadsheet or text delimited spreadsheet
    bool m_bTransaction; // Internal flag to denote status of Transaction

    long m_dCurrentRow; // Index of current row, starting from 1
    long m_dTotalRows; // Total number of rows in spreadsheet
    short m_dTotalColumns; // Total number of columns in Excel spreadsheet.
Largest number of columns in text delimited spreadsheet

    CString m_sSql; // SQL statement to open Excel spreadsheet for reading
    CString m_sDsn; // DSN string to open Excel spreadsheet for reading and
writing
    CString m_stempSql; // Temporary string for SQL statements or for use
by functions
    CString m_stempString; // Temporary string for use by functions
    CString m_sSheetName; // Sheet name of Excel spreadsheet
    CString m_sExcelDriver; // Name of Excel Driver
    CString m_sFile; // Spreadsheet file name
    CString m_sSeparator; // Separator in text delimited spreadsheet
    CString m_sLastError; // Last error message

    CStringArray m_atempArray; // Temporary array for use by functions
    CStringArray m_aFieldNames; // Header row in spreadsheet
    CStringArray m_aRows; // Content of all the rows in spreadsheet

```

```

        CDatabase *m_Database; // Database variable for Excel spreadsheet
        CRecordset *m_rSheet; // Recordset for Excel spreadsheet
};

// Open spreadsheet for reading and writing
CSpreadSheet::CSpreadSheet(CString File, CString SheetOrSeparator, bool
Backup) :
m_Database(NULL), m_rSheet(NULL), m_sFile(File),
m_dTotalRows(0), m_dTotalColumns(0), m_dCurrentRow(1),
m_bAppend(false), m_bBackup(Backup), m_bTransaction(false)
{
    // Detect whether file is an Excel spreadsheet or a text delimited file
    m_stempString = m_sFile.Right(4);
    m_stempString.MakeLower();
    if (m_stempString == ".xls") // File is an Excel spreadsheet
    {
        m_bExcel = true;
        m_sSheetName = SheetOrSeparator;
        m_sSeparator = ",;?.?";
    }
    else // File is a text delimited file
    {
        m_bExcel = false;
        m_sSeparator = SheetOrSeparator;
    }

    if (m_bExcel) // If file is an Excel spreadsheet
    {
        m_Database = new CDatabase;
        GetExcelDriver();

        m_sDsn.Format("DRIVER={%s};DSN='';FIRSTROWHASNAMES=1;READONLY=FALSE;CRE
ATE_DB=\"%s\";DBQ=%s", m_sExcelDriver, m_sFile, m_sFile);

        if (Open())
        {
            if (m_bBackup)
            {
                if ((m_bBackup) && (m_bAppend))
                {
                    CString tempSheetName = m_sSheetName;
                    m_sSheetName = "CSpreadSheetBackup";
                    m_bAppend = false;
                    if (!Commit())
                    {
                        m_bBackup = false;
                    }
                    m_bAppend = true;
                    m_sSheetName = tempSheetName;
                    m_dCurrentRow = 1;
                }
            }
        }
    }
    else // if file is a text delimited file
    {
        if (Open())

```

```

        {
            if ((m_bBackup) && (m_bAppend))
            {
                m_stempString = m_sFile;
                m_stempSql.Format("%s.bak", m_sFile);
                m_sFile = m_stempSql;
                if (!Commit())
                {
                    m_bBackup = false;
                }
                m_sFile = m_stempString;
            }
        }
    }

// Perform some cleanup functions
CSpreadSheet::~CSpreadSheet()
{
    if (m_Database != NULL)
    {
        m_Database->Close();
        delete m_Database;
    }
}

// Add header row to spreadsheet
bool CSpreadSheet::AddHeaders(CStringArray &FieldNames, bool replace)
{
    if (m_bAppend) // Append to old Sheet
    {
        if (replace) // Replacing header row rather than adding new
columns
        {
            if (!AddRow(FieldNames, 1, true))
            {
                return false;
            }
            else
            {
                return true;
            }
        }

        if (ReadRow(m_atempArray, 1)) // Add new columns
        {
            if (m_bExcel)
            {
                // Check for duplicate header row field
                for (int i = 0; i < FieldNames.GetSize(); i++)
                {
                    for (int j = 0; j < m_atempArray.GetSize();
j++)
                    {
                        if (FieldNames.GetAt(i) ==
m_atempArray.GetAt(j))
                            {

```

```

                                m_sLastError.Format("Duplicate
header row field:%s\n", FieldNames.GetAt(i));
                                return false;
                                }
                                }
                                }

m_atempArray.Append(FieldNames);
if (!AddRow(m_atempArray, 1, true))
{
    m_sLastError = "Problems with adding headers\n";
    return false;
}

// Update largest number of columns if necessary
if (m_atempArray.GetSize() > m_dTotalColumns)
{
    m_dTotalColumns = m_atempArray.GetSize();
}
return true;
}
return false;
}
else // New Sheet
{
    m_dTotalColumns = FieldNames.GetSize();
    if (!AddRow(FieldNames, 1, true))
    {
        return false;
    }
    else
    {
        m_dTotalRows = 1;
        return true;
    }
}
}

// Clear text delimited file content
bool CSpreadSheet::DeleteSheet()
{
    if (m_bExcel)
    {
        if (DeleteSheet(m_sSheetName))
        {
            return true;
        }
        else
        {
            m_sLastError = "Error deleting sheet\n";
            return false;
        }
    }
    else
    {
        m_aRows.RemoveAll();
    }
}

```

```

        m_aFieldNames.RemoveAll();
        m_dTotalColumns = 0;
        m_dTotalRows = 0;
        if (!m_bTransaction)
        {
            Commit();
        }
        m_bAppend = false; // Set flag to new sheet
        return true;
    }
}

// Clear entire Excel spreadsheet content. The sheet itself is not deleted
bool CSpreadSheet::DeleteSheet(CString SheetName)
{
    if (m_bExcel) // If file is an Excel spreadsheet
    {
        // Delete sheet
        m_Database->OpenEx(m_sDsn, CDatabase::noOdbcDialog);
        SheetName = "[" + SheetName + "$A1:IV65536]";
        m_stempSql.Format ("DROP TABLE %s", SheetName);
        try
        {
            m_Database->ExecuteSQL(m_stempSql);
            m_Database->Close();
            m_aRows.RemoveAll();
            m_aFieldNames.RemoveAll();
            m_dTotalColumns = 0;
            m_dTotalRows = 0;
        }
        catch(CDBException *e)
        {
            m_sLastError = e->m_strError;
            m_Database->Close();
            return false;
        }
        return true;
    }
    else // if file is a text delimited file
    {
        return DeleteSheet();
    }
}

// Insert or replace a row into spreadsheet.
// Default is add new row.
bool CSpreadSheet::AddRow(CStringArray &RowValues, long row, bool replace)
{
    long tempRow;

    if (row == 1)
    {
        if (m_bExcel)
        {
            // Check for duplicate header row field for Excel
            spreadsheet
            for (int i = 0; i < RowValues.GetSize(); i++)

```

```

        {
            for (int j = 0; j < RowValues.GetSize(); j++)
            {
                if ((i != j) && (RowValues.GetAt(i) ==
RowValues.GetAt(j)))
                    {
                        m_sLastError.Format("Duplicate header row
field:%s\n", RowValues.GetAt(i));
                        return false;
                    }
            }

            // Check for reduced header row columns
            if (RowValues.GetSize() < m_dTotalColumns)
            {
                m_sLastError = "Number of columns in new header row
cannot be less than the number of columns in previous header row";
                return false;
            }
            m_dTotalColumns = RowValues.GetSize();

            // Update header row
            m_aFieldNames.RemoveAll();
            m_aFieldNames.Copy(RowValues);
        }
        else
        {
            if (m_bExcel)
            {
                if (m_dTotalColumns == 0)
                {
                    m_sLastError = "No header row. Add header row
first\n";
                    return false;
                }
            }

            if (m_bExcel) // For Excel spreadsheet
            {
                if (RowValues.GetSize() > m_aFieldNames.GetSize())
                {
                    m_sLastError = "Number of columns to be added cannot be
greater than the number of fields\n";
                    return false;
                }
            }
            else // For text delimited spreadsheet
            {
                // Update largest number of columns if necessary
                if (RowValues.GetSize() > m_dTotalColumns)
                {
                    m_dTotalColumns = RowValues.GetSize();
                }
            }
        }
    }

```

```

// Convert row values
m_stempString.Empty();
for (int i = 0; i < RowValues.GetSize(); i++)
{
    if (i != RowValues.GetSize()-1) // Not last column
    {
        m_stempSql.Format("\">%s\">%s", RowValues.GetAt(i),
m_sSeparator);
        m_stempString += m_stempSql;
    }
    else // Last column
    {
        m_stempSql.Format("\">%s\\"", RowValues.GetAt(i));
        m_stempString += m_stempSql;
    }
}

if (row)
{
    if (row <= m_dTotalRows) // Not adding new rows
    {
        if (replace) // Replacing row
        {
            m_aRows.SetAt(row-1, m_stempString);
        }
        else // Inserting row
        {
            m_aRows.InsertAt(row-1, m_stempString);
            m_dTotalRows++;
        }

        if (!m_bTransaction)
        {
            Commit();
        }
        return true;
    }
    else // Adding new rows
    {
        // Insert null rows until specified row
        m_dCurrentRow = m_dTotalRows;
        m_stempSql.Empty();
        CString nullString;
        for (int i = 1; i <= m_dTotalColumns; i++)
        {
            if (i != m_dTotalColumns)
            {
                if (m_bExcel)
                {
                    nullString.Format("\"> \"">%s",
m_sSeparator);
                }
                else
                {
                    nullString.Format("\">\">%s",
m_sSeparator);
                }
            }
        }
    }
}

```

```

        m_stempSql += nullString;
    }
    else
    {
        if (m_bExcel)
        {
            m_stempSql += "\" \";
        }
        else
        {
            m_stempSql += "\"\\"";
        }
    }
}
for (int j = m_dTotalRows + 1; j < row; j++)
{
    m_dCurrentRow++;
    m_aRows.Add(m_stempSql);
}
}
else
{
    tempRow = m_dCurrentRow;
    m_dCurrentRow = m_dTotalRows;
}

// Insert new row
m_dCurrentRow++;
m_aRows.Add(m_stempString);

if (row > m_dTotalRows)
{
    m_dTotalRows = row;
}
else if (!row)
{
    m_dTotalRows = m_dCurrentRow;
    m_dCurrentRow = tempRow;
}
if (!m_bTransaction)
{
    Commit();
}
return true;
}

// Replace or add a cell into Excel spreadsheet using header row or column
alphabet.
// Default is add cell into new row.
// Set Auto to false if want to force column to be used as header name
bool CSpreadSheet::AddCell(CString CellValue, CString column, long row, bool
Auto)
{
    short columnIndex = CalculateColumnNumber(column, Auto);
    if (columnIndex == 0)
    {

```

```

        return false;
    }

    if (AddCell(CellValue, columnIndex, row))
    {
        return true;
    }
    return false;
}

// Replace or add a cell into spreadsheet using column number
// Default is add cell into new row.
bool CSpreadSheet::AddCell(CString CellValue, short column, long row)
{
    if (column == 0)
    {
        m_sLastError = "Column cannot be zero\n";
        return false;
    }

    long tempRow;

    if (m_bExcel) // For Excel spreadsheet
    {
        if (column > m_aFieldNames.GetSize() + 1)
        {
            m_sLastError = "Cell column to be added cannot be greater
than the number of fields\n";
            return false;
        }
    }
    else // For text delimited spreadsheet
    {
        // Update largest number of columns if necessary
        if (column > m_dTotalColumns)
        {
            m_dTotalColumns = column;
        }
    }

    if (row)
    {
        if (row <= m_dTotalRows)
        {
            ReadRow(m_atempArray, row);

            // Change desired row
            m_atempArray.SetAtGrow(column-1, CellValue);

            if (row == 1)
            {
                if (m_bExcel) // Check for duplicate header row field
                {
                    for (int i = 0; i < m_atempArray.GetSize();
i++)
                    {

```

```

        for (int j = 0; j <
m_atempArray.GetSize(); j++)
        {
            if ((i != j) &&
(m_atempArray.GetAt(i) == m_atempArray.GetAt(j)))
            {
                m_sLastError.Format("Duplicate header row field:%s\n",
m_atempArray.GetAt(i));
                return false;
            }
        }
    }
}

// Update header row
m_aFieldNames.RemoveAll();
m_aFieldNames.Copy(m_atempArray);
}

if (!AddRow(m_atempArray, row, true))
{
    return false;
}

if (!m_bTransaction)
{
    Commit();
}
return true;
}
else
{
    // Insert null rows until specified row
    m_dCurrentRow = m_dTotalRows;
    m_stempSql.Empty();
    CString nullString;
    for (int i = 1; i <= m_dTotalColumns; i++)
    {
        if (i != m_dTotalColumns)
        {
            if (m_bExcel)
            {
                nullString.Format("\" \"\ \"%s",
m_sSeparator);
            }
            else
            {
                nullString.Format("\"\" \"%s",
m_sSeparator);
            }
            m_stempSql += nullString;
        }
        else
        {
            if (m_bExcel)
            {

```

```

        m_stempSql += "\" \";
    }
    else
    {
        m_stempSql += "\"\"";
    }
}
for (int j = m_dTotalRows + 1; j < row; j++)
{
    m_dCurrentRow++;
    m_aRows.Add(m_stempSql);
}
}
else
{
    tempRow = m_dCurrentRow;
    m_dCurrentRow = m_dTotalRows;
}

// Insert cell
m_dCurrentRow++;
m_stempString.Empty();
for (int j = 1; j <= m_dTotalColumns; j++)
{
    if (j != m_dTotalColumns) // Not last column
    {
        if (j != column)
        {
            if (m_bExcel)
            {
                m_stempSql.Format("\" \">%s", m_sSeparator);
            }
            else
            {
                m_stempSql.Format("\"\"">%s", m_sSeparator);
            }
            m_stempString += m_stempSql;
        }
        else
        {
            m_stempSql.Format("\">%s">%s", CellValue,
m_sSeparator);
            m_stempString += m_stempSql;
        }
    }
    else // Last column
    {
        if (j != column)
        {
            if (m_bExcel)
            {
                m_stempString += "\" \";
            }
            else
            {

```

```

        m_stempString += "\\\"";
    }
}
else
{
    m_stempSql.Format("%s\"", CellValue);
    m_stempString += m_stempSql;
}
}
}

m_aRows.Add(m_stempString);

if (row > m_dTotalRows)
{
    m_dTotalRows = row;
}
else if (!row)
{
    m_dTotalRows = m_dCurrentRow;
    m_dCurrentRow = tempRow;
}
if (!m_bTransaction)
{
    Commit();
}
return true;
}

// Search and replace rows in Excel spreadsheet
bool CSpreadSheet::ReplaceRows(CStringArray &NewRowValues, CStringArray
&OldRowValues)
{
    if (m_bExcel) // If file is an Excel spreadsheet
    {
        m_Database->OpenEx(m_sDsn, CDatabase::noOdbcDialog);
        m_stempSql.Format("UPDATE [%s] SET ", m_sSheetName);
        for (int i = 0; i < NewRowValues.GetSize(); i++)
        {
            m_stempString.Format("[%s]='%s', ", m_aFieldNames.GetAt(i),
NewRowValues.GetAt(i));
            m_stempSql = m_stempSql + m_stempString;
        }
        m_stempSql.Delete(m_stempSql.GetLength()-2, 2);
        m_stempSql = m_stempSql + " WHERE (";
        for (int j = 0; j < OldRowValues.GetSize()-1; j++)
        {
            m_stempString.Format("[%s]='%s' AND ",
m_aFieldNames.GetAt(j), OldRowValues.GetAt(j));
            m_stempSql = m_stempSql + m_stempString;
        }
        m_stempSql.Delete(m_stempSql.GetLength()-4, 5);
        m_stempSql += ")";

        try
        {
            m_Database->ExecuteSQL(m_stempSql);

```

```

        m_Database->Close();
        Open();
        return true;
    }
    catch(CDBException *e)
    {
        m_sLastError = e->m_strError;
        m_Database->Close();
        return false;
    }
}
else // if file is a text delimited file
{
    m_sLastError = "Function not available for text delimited
file\n";
    return false;
}
}

// Read a row from spreadsheet.
// Default is read the next row
bool CSpreadSheet::ReadRow(CStringArray &RowValues, long row)
{
    // Check if row entered is more than number of rows in sheet
    if (row <= m_aRows.GetSize())
    {
        if (row != 0)
        {
            m_dCurrentRow = row;
        }
        else if (m_dCurrentRow > m_aRows.GetSize())
        {
            return false;
        }
        // Read the desired row
        RowValues.RemoveAll();
        m_stempString = m_aRows.GetAt(m_dCurrentRow-1);
        m_dCurrentRow++;

        // Search for separator to split row
        int separatorPosition;
        m_stempSql.Format("\">%s\\"", m_sSeparator);
        separatorPosition = m_stempString.Find(m_stempSql); // If
separator is "?"
        if (separatorPosition != -1)
        {
            // Save columns
            int nCount = 0;
            int stringStartingPosition = 0;
            while (separatorPosition != -1)
            {
                nCount = separatorPosition - stringStartingPosition;

                RowValues.Add(m_stempString.Mid(stringStartingPosition, nCount));
                stringStartingPosition = separatorPosition +
m_stempSql.GetLength();
            }
        }
    }
}

```

```

        separatorPosition = m_stempString.Find(m_stempSql,
stringStartingPosition);
    }
    nCount = m_stempString.GetLength() -
stringStartingPosition;
    RowValues.Add(m_stempString.Mid(stringStartingPosition,
nCount));

    // Remove quotes from first column
    m_stempString = RowValues.GetAt(0);
    m_stempString.Delete(0, 1);
    RowValues.SetAt(0, m_stempString);

    // Remove quotes from last column
    m_stempString = RowValues.GetAt(RowValues.GetSize()-1);
    m_stempString.Delete(m_stempString.GetLength()-1, 1);
    RowValues.SetAt(RowValues.GetSize()-1, m_stempString);

    return true;
}
else
{
    // Save columns
    separatorPosition = m_stempString.Find(m_sSeparator); // if
separator is ?
    if (separatorPosition != -1)
    {
        int nCount = 0;
        int stringStartingPosition = 0;
        while (separatorPosition != -1)
        {
            nCount = separatorPosition -
stringStartingPosition;

            RowValues.Add(m_stempString.Mid(stringStartingPosition, nCount));
            stringStartingPosition = separatorPosition +
m_sSeparator.GetLength();
            separatorPosition =
m_stempString.Find(m_sSeparator, stringStartingPosition);
        }
        nCount = m_stempString.GetLength() -
stringStartingPosition;

        RowValues.Add(m_stempString.Mid(stringStartingPosition, nCount));
        return true;
    }
    else // Treat spreadsheet as having one column
    {
        // Remove opening and ending quotes if any
        int quoteBegPos = m_stempString.Find('\\"');
        int quoteEndPos = m_stempString.ReverseFind('\\"');
        if ((quoteBegPos == 0) && (quoteEndPos ==
m_stempString.GetLength()-1))
        {
            m_stempString.Delete(0, 1);
            m_stempString.Delete(m_stempString.GetLength()-
1, 1);

```

```

        }

        RowValues.Add(m_stempString);
    }
}

m_sLastError = "Desired row is greater than total number of rows in
spreadsheet\n";
return false;
}

// Read a column from Excel spreadsheet using header row or column alphabet.
// Set Auto to false if want to force column to be used as header name
bool CSpreadSheet::ReadColumn(CStringArray &ColumnValues, CString column,
bool Auto)
{
    short columnIndex = CalculateColumnNumber(column, Auto);
    if (columnIndex == 0)
    {
        return false;
    }

    if (ReadColumn(ColumnValues, columnIndex))
    {
        return true;
    }
    return false;
}

// Read a column from spreadsheet using column number
bool CSpreadSheet::ReadColumn(CStringArray &ColumnValues, short column)
{
    if (column == 0)
    {
        m_sLastError = "Column cannot be zero\n";
        return false;
    }

    int tempRow = m_dCurrentRow;
    m_dCurrentRow = 1;
    ColumnValues.RemoveAll();
    for (int i = 1; i <= m_aRows.GetSize(); i++)
    {
        // Read each row
        if (ReadRow(m_atempArray, i))
        {
            // Get value of cell in desired column
            if (column <= m_atempArray.GetSize())
            {
                ColumnValues.Add(m_atempArray.GetAt(column-1));
            }
            else
            {
                ColumnValues.Add("");
            }
        }
    }
    else
}

```

```

        {
            m_dCurrentRow = tempRow;
            m_sLastError = "Error reading row\n";
            return false;
        }
    }
    m_dCurrentRow = tempRow;
    return true;
}

// Read a cell from Excel spreadsheet using header row or column alphabet.
// Default is read the next cell in next row.
// Set Auto to false if want to force column to be used as header name
bool CSpreadSheet::ReadCell (CString &CellValue, CString column, long row,
bool Auto)
{
    short columnIndex = CalculateColumnNumber(column, Auto);
    if (columnIndex == 0)
    {
        return false;
    }

    if (ReadCell(CellValue, columnIndex, row))
    {
        return true;
    }
    return false;
}

// Read a cell from spreadsheet using column number.
// Default is read the next cell in next row.
bool CSpreadSheet::ReadCell (CString &CellValue, short column, long row)
{
    if (column == 0)
    {
        m_sLastError = "Column cannot be zero\n";
        return false;
    }

    int tempRow = m_dCurrentRow;
    if (row)
    {
        m_dCurrentRow = row;
    }
    if (ReadRow(m_atempArray, m_dCurrentRow))
    {
        // Get value of cell in desired column
        if (column <= m_atempArray.GetSize())
        {
            CellValue = m_atempArray.GetAt(column-1);
        }
        else
        {
            CellValue.Empty();
            m_dCurrentRow = tempRow;
            return false;
        }
    }
}

```

```

        m_dCurrentRow = tempRow;
        return true;
    }
    m_dCurrentRow = tempRow;
    m_sLastError = "Error reading row\n";
    return false;
}

// Begin transaction
void CSpreadSheet::BeginTransaction()
{
    m_bTransaction = true;
}

// Save changes to spreadsheet
bool CSpreadSheet::Commit()
{
    if (m_bExcel) // If file is an Excel spreadsheet
    {
        m_Database->OpenEx(m_sDsn, CDatabase::noOdbcDialog);

        if (m_bAppend)
        {
            // Delete old sheet if it exists
            m_stempString= "[" + m_sSheetName + "$A1:IV65536]";
            m_stempSql.Format ("DROP TABLE %s", m_stempString);
            try
            {
                m_Database->ExecuteSQL(m_stempSql);
            }
            catch(CDBException *e)
            {
                m_sLastError = e->m_strError;
                m_Database->Close();
                return false;
            }

            // Create new sheet
            m_stempSql.Format("CREATE TABLE [%s$A1:IV65536] (",
m_sSheetName);
            for (int j = 0; j < m_aFieldNames.GetSize(); j++)
            {
                m_stempSql = m_stempSql + "[" +
m_aFieldNames.GetAt(j) +"]" + " char(255), ";
            }
            m_stempSql.Delete(m_stempSql.GetLength()-2, 2);
            m_stempSql += ")";
        }
        else
        {
            // Create new sheet
            m_stempSql.Format("CREATE TABLE [%s] (", m_sSheetName);
            for (int i = 0; i < m_aFieldNames.GetSize(); i++)
            {
                m_stempSql = m_stempSql + "[" +
m_aFieldNames.GetAt(i) +"]" + " char(255), ";
            }

```

```

        m_stempSql.Delete(m_stempSql.GetLength()-2, 2);
        m_stempSql += ")";
    }

    try
    {
        m_Database->ExecuteSQL(m_stempSql);
        if (!m_bAppend)
        {
            m_dTotalColumns = m_aFieldNames.GetSize();
            m_bAppend = true;
        }
    }
    catch(CDBException *e)
    {
        m_sLastError = e->m_strError;
        m_Database->Close();
        return false;
    }

    // Save changed data
    for (int k = 1; k < m_dTotalRows; k++)
    {
        ReadRow(m_atempArray, k+1);

        // Create Insert SQL
        m_stempSql.Format("INSERT INTO [%s$A1:IV%d] (",
m_sSheetName, k);
        for (int i = 0; i < m_atempArray.GetSize(); i++)
        {
            m_stempString.Format("[%s], ",
m_aFieldNames.GetAt(i));
            m_stempSql = m_stempSql + m_stempString;
        }
        m_stempSql.Delete(m_stempSql.GetLength()-2, 2);
        m_stempSql += ") VALUES (";
        for (int j = 0; j < m_atempArray.GetSize(); j++)
        {
            m_stempString.Format("'%'s', ",
m_atempArray.GetAt(j));
            m_stempSql = m_stempSql + m_stempString;
        }
        m_stempSql.Delete(m_stempSql.GetLength()-2, 2);
        m_stempSql += ")";

        // Add row
        try
        {
            m_Database->ExecuteSQL(m_stempSql);
        }
        catch(CDBException *e)
        {
            m_sLastError = e->m_strError;
            m_Database->Close();
            return false;
        }
    }
}

```

```

        m_Database->Close();
        m_bTransaction = false;
        return true;
    }
    else // if file is a text delimited file
    {
        try
        {
            CFile *File = NULL;
            File = new CFile(m_sFile, CFile::modeCreate |
CFile::modeWrite | CFile::shareDenyNone);
            if (File != NULL)
            {
                CArchive *Archive = NULL;
                Archive = new CArchive(File, CArchive::store);
                if (Archive != NULL)
                {
                    for (int i = 0; i < m_aRows.GetSize(); i++)
                    {
                        Archive->WriteString(m_aRows.GetAt(i));
                        Archive->WriteString("\r\n");
                    }
                    delete Archive;
                    delete File;
                    m_bTransaction = false;
                    return true;
                }
                delete File;
            }
        }
        catch(...)
        {
        }
        m_sLastError = "Error writing file\n";
        return false;
    }
}

// Undo changes to spreadsheet
bool CSpreadSheet::RollBack()
{
    if (Open())
    {
        m_bTransaction = false;
        return true;
    }
    m_sLastError = "Error in returning to previous state\n";
    return false;
}

bool CSpreadSheet::Convert(CString SheetOrSeparator)
{
    // Prepare file
    m_stempString = m_sFile;
    m_stempString.Delete(m_stempString.GetLength()-4, 4);
    if (m_bExcel) // If file is an Excel spreadsheet
    {

```

```

m_stempString += ".csv";
CSpreadSheet tempSheet(m_stempString, SheetOrSeparator, false);

// Stop convert if text delimited file exists
if (tempSheet.GetTotalColumns() != 0)
{
    return false;
}

tempSheet.BeginTransaction();

for (int i = 1; i <= m_dTotalRows; i++)
{
    if (!ReadRow(m_atempArray, i))
    {
        return false;
    }
    if (!tempSheet.AddRow(m_atempArray, i))
    {
        return false;
    }
}
if (!tempSheet.Commit())
{
    return false;
}
return true;
}
else // if file is a text delimited file
{
    m_stempString += ".xls";
    CSpreadSheet tempSheet(m_stempString, SheetOrSeparator, false);

    // Stop convert if Excel file exists
    if (tempSheet.GetTotalColumns() != 0)
    {
        return false;
    }

    GetFieldNames(m_atempArray);

    // Check for duplicate header row field
    bool duplicate = false;
    for (int i = 0; i < m_atempArray.GetSize(); i++)
    {
        for (int j = 0; j < m_atempArray.GetSize(); j++)
        {
            if ((i != j) && (m_atempArray.GetAt(i) ==
m_atempArray.GetAt(j)))
            {
                m_sLastError.Format("Duplicate header row
field:%s\n", m_atempArray.GetAt(i));
                duplicate = true;
            }
        }
    }
}

```

```

if (duplicate) // Create dummy header row
{
    m_atempArray.RemoveAll();
    for (int k = 1; k <= m_dTotalColumns; k++)
    {
        m_stempString.Format("%d", k);
        m_atempArray.Add(m_stempString);
    }

    if (!tempSheet.AddHeaders(m_atempArray))
    {
        return false;
    }

    for (int l = 1; l <= m_dTotalRows; l++)
    {
        if (!ReadRow(m_atempArray, l))
        {
            return false;
        }
        if (!tempSheet.AddRow(m_atempArray, l+1))
        {
            return false;
        }
    }
    return true;
}
else
{
    if (!tempSheet.AddHeaders(m_atempArray))
    {
        return false;
    }

    for (int l = 2; l <= m_dTotalRows; l++)
    {
        if (!ReadRow(m_atempArray, l))
        {
            return false;
        }
        if (!tempSheet.AddRow(m_atempArray, l))
        {
            return false;
        }
    }
    return true;
}
}

// Open a text delimited file for reading or writing
bool CSpreadSheet::Open()
{
    if (m_bExcel) // If file is an Excel spreadsheet
    {
        m_Database->OpenEx(m_sDsn, CDatabase::noOdbcDialog);
    }
}

```

```

// Open Sheet
m_rSheet = new CRecordset( m_Database );
m_sSql.Format("SELECT * FROM [%s$A1:IV65536]", m_sSheetName);
try
{
    m_rSheet->Open(CRecordset::forwardOnly, m_sSql,
CRecordset::readOnly);
}
catch(...)
{
    delete m_rSheet;
    m_rSheet = NULL;
    m_Database->Close();
    return false;
}

// Get number of columns
m_dTotalColumns = m_rSheet->m_nResultCols;

if (m_dTotalColumns != 0)
{
    m_aRows.RemoveAll();
    m_stempString.Empty();
    m_bAppend = true;
    m_dTotalRows++; // Keep count of total number of rows

    // Get field names i.e header row
    for (int i = 0; i < m_dTotalColumns; i++)
    {
        m_stempSql = m_rSheet-
>m_rgODBCFieldInfos[i].m_strName;
        m_aFieldNames.Add(m_stempSql);

        // Join up all the columns into a string
        if (i != m_dTotalColumns-1) // Not last column
        {
            m_stempString = m_stempString + "\"\" +
m_stempSql + "\"\" + m_sSeparator;
        }
        else // Last column
        {
            m_stempString = m_stempString + "\"\" +
m_stempSql + "\"\";
        }
    }

    // Store the header row as the first row in memory
    m_aRows.Add(m_stempString);

    // Read and store the rest of the rows in memory
    while (!m_rSheet->IsEOF())
    {
        m_dTotalRows++; // Keep count of total number of rows
        try
        {
            // Get all the columns in a row
            m_stempString.Empty();

```

```

        for (short column = 0; column <
m_dTotalColumns; column++)
        {
            m_rSheet->GetFieldValue(column,
m_stempSql);

            // Join up all the columns into a string
            if (column != m_dTotalColumns-1) // Not
last column
            {
                m_stempString = m_stempString +
"\\"" + m_stempSql + "\"\" + m_sSeparator;
            }
            else // Last column
            {
                m_stempString = m_stempString +
"\\"" + m_stempSql + "\"\";
            }

            // Store the obtained row in memory
            m_aRows.Add(m_stempString);
            m_rSheet->MoveNext();
        }
        catch (...)
        {
            m_sLastError = "Error reading row\n";
            delete m_rSheet;
            m_rSheet = NULL;
            m_Database->Close();
            return false;
        }
    }

    m_rSheet->Close();
    delete m_rSheet;
    m_rSheet = NULL;
    m_Database->Close();
    m_dCurrentRow = 1;
    return true;
}
else // if file is a text delimited file
{
    try
    {
        CFile *File = NULL;
        File = new CFile(m_sFile, CFile::modeRead |
CFile::shareDenyNone);
        if (File != NULL)
        {
            CArchive *Archive = NULL;
            Archive = new CArchive(File, CArchive::load);
            if (Archive != NULL)
            {
                m_aRows.RemoveAll();
                // Read and store all rows in memory

```

```

        while(Archive->ReadString(m_stempString))
        {
            m_aRows.Add(m_stempString);
        }
        ReadRow(m_aFieldNames, 1); // Get field names
        delete Archive;
        delete File;

        // Get total number of rows
        m_dTotalRows = m_aRows.GetSize();

        // Get the largest number of columns
        for (int i = 0; i < m_aRows.GetSize(); i++)
        {
            ReadRow(m_atempArray, i);
            if (m_atempArray.GetSize() >
                m_dTotalColumns)
            {
                m_dTotalColumns =
                    m_atempArray.GetSize();
            }

            if (m_dTotalColumns != 0)
            {
                m_bAppend = true;
            }
            return true;
        }
        delete File;
    }
}
catch(...)
{
}
m_sLastError = "Error in opening file\n";
return false;
}

// Convert Excel column in alphabet into column number
short CSpreadSheet::CalculateColumnNumber(CString column, bool Auto)
{
    if (Auto)
    {
        int firstLetter, secondLetter;
        column.MakeUpper();

        if (column.GetLength() == 1)
        {
            firstLetter = column.GetAt(0);
            return (firstLetter - 65 + 1); // 65 is A in ascii
        }
        else if (column.GetLength() == 2)
        {
            firstLetter = column.GetAt(0);

```

```

                secondLetter = column.GetAt(1);
                return ((firstLetter - 65 + 1)*26 + (secondLetter - 65 +
1)); // 65 is A in ascii
            }
        }

        // Check if it is a valid field name
        for (int i = 0; i < m_aFieldNames.GetSize(); i++)
        {
            if (!column.Compare(m_aFieldNames.GetAt(i)))
            {
                return (i + 1);
            }
        }
        m_sLastError = "Invalid field name or column alphabet\n";
        return 0;
    }

    // Get the name of the Excel-ODBC driver
    void CSpreadSheet::GetExcelDriver()
    {
        char szBuf[2001];
        WORD cbBufMax = 2000;
        WORD cbBufOut;
        char *pszBuf = szBuf;

        // Get the names of the installed drivers ("odbcinst.h" has to be
included )
        if(!SQLGetInstalledDrivers(szBuf,cbBufMax,& cbBufOut))
        {
            m_sExcelDriver = "";
        }

        // Search for the driver...
        do
        {
            if( strstr( pszBuf, "Excel" ) != 0 )
            {
                // Found !
                m_sExcelDriver = CString( pszBuf );
                break;
            }
            pszBuf = strchr( pszBuf, '\0' ) + 1;
        }
        while( pszBuf[1] != '\0' );
    }

#endif

```

Appendix L: Computer Code

SN.h

```
#pragma once

#include <afxdb.h>

using namespace System;
using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Windows::Forms;
using namespace System::Data;
using namespace System::Drawing;

namespace Saunternot {

    /// <summary>
    /// Summary for SN
    ///
    /// WARNING: If you change the name of this class, you will need to
    change the
    ///
    'Resource File Name' property for the managed resource
    compiler tool
    ///
    associated with all .resx files this class depends on.
    Otherwise,
    ///
    the designers will not be able to interact properly with
    localized
    ///
    resources associated with this form.
    /// </summary>
    public ref class SN : public System::Windows::Forms::Form
    {
    public:
        SN(void)
        {
            InitializeComponent();
            //
            //TODO: Add the constructor code here
            //
        }

    protected:
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        ~SN()
        {
            if (components)
            {
                delete components;
            }
        }

    public: System::Windows::Forms::Button^ button1;
    public: System::Windows::Forms::Label^ label1;
    
```

```

protected:

private:
    /// <summary>
    /// Required designer variable.
    /// </summary>
    System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    void InitializeComponent(void)
    {
        System::ComponentModel::ComponentResourceManager^
resources = (gcnew
System::ComponentModel::ComponentResourceManager(SN::typeid));
        this->button1 = (gcnew System::Windows::Forms::Button());
        this->label1 = (gcnew System::Windows::Forms::Label());
        this->SuspendLayout();
        //
        // button1
        //
        this->button1->DialogResult =
System::Windows::Forms::DialogResult::OK;
        this->button1->Location = System::Drawing::Point(185, 106);
        this->button1->Name = L"button1";
        this->button1->RightToLeft =
System::Windows::Forms::RightToLeft::No;
        this->button1->Size = System::Drawing::Size(92, 30);
        this->button1->TabIndex = 0;
        this->button1->Text = L"OK";
        this->button1->UseVisualStyleBackColor = true;
        //
        // label1
        //
        this->label1->Location = System::Drawing::Point(12, 9);
        this->label1->Name = L"label1";
        this->label1->Size = System::Drawing::Size(438, 94);
        this->label1->TabIndex = 1;
        this->label1->TextAlign =
System::Drawing::ContentAlignment::MiddleCenter;
        //
        // SN
        //
        this->AcceptButton = this->button1;
        this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
        this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
        this->AutoValidate =
System::Windows::Forms::AutoValidate::EnableAllowFocusChange;
        this->ClientSize = System::Drawing::Size(462, 148);
        this->Controls->Add(this->button1);
        this->Controls->Add(this->label1);
        this->Enabled = false;
    }

```

```
        this->Icon = (cli::safe_cast<System::Drawing::Icon^
>(resources->GetObject(L"$this.Icon")));
        this->MaximizeBox = false;
        this->MinimizeBox = false;
        this->Name = L"SN";
        this->Text = L"WARNING";
        this->TopMost = true;
        this->ResumeLayout(false);

    }
#pragma endregion

};

}
```

Datasheets

PIC18F2520 Microcontroller

<http://ww1.microchip.com/downloads/en/DeviceDoc/39631D.pdf>

TLP434A/RLP434A

http://www.datasheetcatalog.org/datasheets/37/467327_DS.pdf

Boost Converter

<http://www.datasheetcatalog.org/datasheet/texasinstruments/ucc2941-5.pdf>

Spartan XC3S200 Datasheet

<http://www.tfh-berlin.de/~liebmann/Spartan3ds099.pdf>

Spartan 3 User Guide

http://www.digilentinc.com/Data/Products/S3BOARD/S3BOARD_RM.pdf

Bill of Materials

Design III

Component	Vendor	Part #	Value	Quantity	Cost
Spartan3 XC3S200	Digilent	XC3S1200E Spartan3E		1	\$99.00 ⁴
RF Link - 2400bps - 434MHz	SparkFun Electronics	WRL-00872		1	\$13.95
2 "AAA" Battery Holder	RadioShack	270-398		1	\$0.99
Boost Converter	DigiKey	UCC3941N-5		1	\$4.05
Potentiometer	Unicorn Electronics	11-0240	50k Ω	1	\$2.90
PIC	DigiKey	PIC18F252		1	\$9.25 ⁵
Capacitor			22 μ F	1	
Capacitor			10 μ F	1	
Capacitor			75 μ F	1	
Resistor			13.3k Ω	1	

⁴ \$99.00 cost was not incurred by group as a Spartan3 board was already in our possession.

⁵ \$9.25 cost was not incurred by group as the ECE lab had the PIC18F252 on hand.

Works Cited

- [1] Pandya, S. (2001, February). *Nursing Homes*. Retrieved September 26, 2007, from AARP: <http://www.aarp.org/research/longtermcare/nursinghomes/aresearch-import-669-FS10R.html>
- [2] Moody, E. F. (2007). *Nursing Home Statistics (AHCA)*. Retrieved September 26, 2007, from <http://www.efmoody.com/longterm/nursingstatistics.html>
- [3] DeParle, N.-A. (2000, July 27). *Testimony on Nursing Home Staffing*. Retrieved September 26, 2007, from Assistant Secretary for Legislation (ASL): Department of Health and Human Services: <http://www.hhs.gov/asl/testify/t000727a.html>
- [4] Stanley Logistics, Inc. (2004). *Senior Technologies*, a division of Stanley Security Solutions, Inc., from <http://www.seniortechnologies.com>
- [5] J. Spencer, B. Frizzelle, P. Page, J. Volger, *Global Positioning System*. Blackwell Publishing Ltd., 2003, p. 27
- [6] *Trilateration*. (2007, May). Retrieved September 24, 2007, from Wikipedia®: <http://en.wikipedia.org/wiki/Image:Trilateration.svg>
- [7] C. Nerguizian, C. Despins, S. Affes, *A Framework for Indoor Geolocation using an Intelligent System*. 3rd WLAN Workshop, 2001, INRS Telecommunications.
- [8] *How do motion sensing lights and burglar alarms work?* (2007). Retrieved September 26, 2007, from How Stuff Works: <http://computer.howstuffworks.com/question238.htm>
- [9] *How Infrared motion detector components work*. (2007, August). Retrieved September 26, 2007, from Glolab: <http://www.glolab.com/pirparts/infrared.html>
- [10] *Part 1: Active and Pasive RFID: Two Distinct, But Complementary, Technologies for Real-Time Supply Chain Visibility*. (2002). Retrieved September 29, 2007, from www.autoid.org/2002_Documents/sc31_wg4/docs_501-520/520_18000-7_WhitePaper.pdf -
- [11] Sorrells, P. (1998). *Passive RFID Basics*. Retrieved October 1, 2007, from Microchip: ww1.microchip.com/downloads/en/AppNotes/00680b.pdf
- [12] Marsh, M. (2007 , September). *Magnetic coupled transponder systems*. Retrieved October 2, 2007, from Transponder News: <http://rapidttp.com/transponder/magnetic.html>
- [13] Marsh, M. (2007, September). *Electric coupled transponder systems*. Retrieved October 2, 2007, from Transponder News: <http://rapidttp.com/transponder/electric.html>

- [14] Dyna-Sys. *TI-RFID*. (2005). Retrieved January 20, 2008, from RFIDUSA:
<http://www.rfidusa.com/>
- [15] Laynetworks.com. (2007). *Slotted Aloha - Aloha Protocol - Free Computer Science Tutorials - Provided by Laynetworks.com*. Retrieved February 16, 2008, from Laynetworks.com: <http://www.laynetworks.com/Slotted%20Aloha.htm>
- [16] Holtek Semiconductor. (2003). *Holtek Semiconductor HT12A/HT12E -- 2¹² Series of Encoders*. Retrieved 3 11, 2008, from Holtek Semiconductor:
http://www.holtek.com/pdf/consumer/2_12ev110.pdf
- [17] Holtek Semiconductor. (2002). *Holtek Semiconductor HT12D/HT12F -- 2¹² Series of Decoders*. Retrieved 3 11, 2008, from Holtek Semiconductor:
http://www.holtek.com/pdf/consumer/2_12dv110.pdf
- [18] *ADA Accessibility Guidelines for Buildings and Facilities (ADAAG)*. (2002, September). Retrieved February 5, 2008, from ADAAG: <http://www.access-board.gov/adaag/html/adaag.htm#4.13>
- [19] Hellie T. *Walking Site*. (2008). Retrieved February 18, 2008, from BellaOnline - Voice of Women: <http://www.bellaonline.com/articles/art20257.asp>
- [20] *RS232 Data Interface: a Tutorial on Data Interface and Cables*. (2008). Retrieved March 30, 2008, from ARC Electronics – a DCE Company:
<http://www.arcelect.com/rs232.htm>